Olympiads in Informatics, 2012, Vol. 6, 115–132



# Type of Questions – The Case of Computer Science

Noa RAGONIS
*School of Education, Computer Science Department, Beit Berl College*
*Department of Education in Technology and Science*
*Technion – Israel Institute of Technology, Israel*
*e-mail: noarag@beitberl.ac.il*

**Abstract.** In this paper, I explore and discuss the variety of types of questions that can be used by computer science educators in different teaching situations and processes: in classroom lessons, in the computer lab, as homework, or in tests. The use of various types of questions offers many advantages, both for learners and for the teaching process. Twelve different types of question are discussed. Each is presented by its classification title, a short description of the specific type of question, a concrete example or an example pattern, and a short pedagogical discussion that includes remarks on cognitive aspects. Three general types of questions are presented and discussed: combination questions, narrative questions, and closed questions. These types of questions relate to "programming-like" assignments since they are the most common subject encountered in the teaching of computer science (Java is used as the implementation language). However, as discussed here in brief, most of the question types are also suitable for most other contents in the teaching of computer science.

**Key words:** computer and information science education, algorithms, human factors, types of questions, cognition in computer science, problem solving.

## 1. Introduction

One of the traditional problem-solving scenarios in computer science starts with the presentation of an open problem described as a "story", continues with its analysis and planning of its solution, and ends with the presentation of a solution as an algorithm in either pseudo-code or in a specific programming language. It is important, however, that computer science educators be aware of the fact that there are many additional types of questions, each of which requires a variety of thinking processes that enable to use a wider spectrum of cognitive skills.

Questions types within the computer science (CS) discipline appear in textbooks and on teaching websites, and are less frequently discussed in the contents of question patterns used to broaden the spectrum of CS educators' teaching tools. This paper aims to present a collection of CS question types as a tool for CS educators in preparing teaching artifacts to be used in different teaching situations and processes: classroom lessons, the computer lab, homework, or tests. The focus is on pedagogical issues intended to expand the educators' perspective with respect to question design. CS educators at all teaching levels, middle schools, high schools and universities, can find this presentation helpful

when planning their teaching process. The presentation of the question types and pattern versions includes recommendations on the use of the questions while teaching.

Three main pedagogical targets can be achieved by integrating different types of questions in the teaching of computer science: (1) they illuminate different aspects of the learned content; (2) they require students to use different cognitive skills; and (3) they enable educators to vary the teaching tools they use. The design of questions that require different cognitive skills is significant, first, in order to present each individual cognitive skill and illuminate different aspects of the learned content; second, since different students express their knowledge differently and it is important to give each of them the opportunity to articulate that knowledge; and third, to develop and enrich the cognitive skills of all students. Furthermore, the use of combinations of different question types throughout the teaching process helps maintain the students' interest, attention, and curiosity.

The examples or example patterns used here to demonstrate each of the question types relate to "programming-like" assignments, since they are the most common subject encountered in the teaching of CS. Yet, most of the question types can be assimilating into any CS content, as will be discussed in Section 6 below, in the context of Automata Theory, for example.

## 2. Background

Most of the discussion on questions and cognition within the teaching process of CS relates to problem-solving strategies and illuminates strategies such as stepwise refinement (e.g., Batory, *et al.*, 2004; Vasconcelos, 2007), algorithmic patterns (e.g., Muller, *et al.*, 2007; Ginat 2009), roles of variables (e.g., Sajaniemi 2005; Sorva, *et al.*, 2007), or tracing strategies (e.g., Venables, *et al.*, 2009). Other discussions relate to strategies adopted by experts versus novices (e.g., Brand-Gruwel, *et al.*, 2005], and other aspects like misleading problem-solving passes such as the design-by-keyword syndrome (Ginat, 2003). I believe that using different types of questions consistently throughout the teaching process enhances learners' skills and abilities with respect to problem-solving processes, since they investigate different analysis tools and expand their ability to examine a problem from multiple points of views.

Another relatively broad discussion in the context of questions in the CS domain is about the evaluation of student artifacts and the progression of learning (e.g., Chamillard and Braun, 2000; Byckling and Sajaniemi, 2006), as well as automatic assessing systems (e.g., Spacco *et al.*, 2006; English and Rosenthal, 2009). In this paper, I will not address assessment aspects but rather the thought processes that arise when students actively solve different types of problems.

Cognition is the process of thought. Cognitive skills are basic mental abilities we all use to think, to study and to learn. Cognitive processes can be analyzed from different perspectives within different contexts. In psychology or philosophy, for example, the concept of cognition is closely related to abstract concepts such as mind, reasoning, perception,

intelligence, learning, and many others that describe the mind's capabilities. The field of cognition focuses on the study of specific mental processes, such as comprehension, inference, decision-making, planning, and learning. Specifically, with respect to CS, we are familiar with the advanced cognitive skills of abstraction, generalization, concretization and meta-reasoning. As mentioned above, it is important to develop the learners' cognitive skills. The use of a variety of question types is one way to achieve this goal. The literature on CS education reveals only a restricted offering of research work related to the cognition aspects of different types of questions. Some research has been done with the intention of adapting Bloom's Taxonomy to the domain of CS. Bloom's Taxonomy was first described in 1956 as a hierarchical model of the cognitive domain (Bloom *et al.*, 1956), followed by several significant changes made by Anderson *et al.* (2001). Thompson *et al.* (2008) reviewed the work done throughout the past decade so as to apply Bloom's taxonomy to CS course design, evaluation, and assessment, and to provide an interpretation of the taxonomy that can be applied to introductory programming exams. This interpretation focused on the cognitive skills involved in addressing several types of questions, whereas Jones *et al.* (2009) focused explicitly on written examinations. These researchers determined the difficulty level of each question in an examination paper based on the criteria of keyword/s found in the question, and presented a cross-analysis across student performance, cognitive skill requirements, and module learning outcomes. Different kinds of studies, that combined computer science questions and cognition, relate to different automata systems aimed at question-answering processes. Pomerantz (2002), for example, evaluated four taxonomies of question types to determine the expressiveness of each for questions received by digital reference services. He offered a faceted classification scheme that can be used as a basis for automating parts of a reference question-answering process. The work of Yang *et al.* (2008) is another example of automated question-generating systems. This research introduced a method of designing a test question database management system based on the three-tier B/S structure, analyzing the features of the test question database structure, and expatiating the grouping algorithmic calculation, focusing on the main control parameters of knowledge points and question-type data. The research results indicate that the application of the system not only improves the work efficiency of teachers, but also positively boosts the teaching reform. Those results support my point of view that emphasizing different types of questions improves the learning-teaching process.

The main aim of the current paper is to broaden CS educators' tools for designing different types of questions. The presentation of each question type focuses on possible ways of using them in teaching processes. Emphasize is on pedagogical approaches and the discussion involves informal cognition considerations.

## 3. Presentation Structure of Question Types

This paper focuses on question patterns and presents twelve types of questions. I suggest several fundamental types and describe several variations for each. Clearly, additional

types of questions, as well as combination of different types of questions, exist and can be developed and used according to need.

Each type of question is presented in the following structure: a title that reflects the ***classification*** of the question type; a short ***description*** of the specific type of question; a concrete ***example*** or ***general pattern*** that demonstrates the said type of question; different ***variations*** of the question type; and a short pedagogical and cognitive ***discussion*** about the said type of question. Since my purpose here is to present a variety of questions in the context of computer science education, most of the examples will be quite simple to solve. For each type of question, it is clearly possible to develop a range of questions on different complexity levels, from both the algorithmic and the cognitive points of view. In addition, within each type of question, additional variations may be presented that require different cognitive skills.

After presenting the twelve question types (Section 4), three global aspects of question types are presented (Section 5), and the assimilation of the different types of questions into different CS contents is discussed (Section 6).

Following are several remarks about the actual use of this collection of questions in the computer science class:

- There is no specific rule for the presentation order of the twelve questions types.
- No specific rule or guidelines can be formulated with respect to the order in which the different types of questions should be used. Each CS educator should select the appropriate type of question and its complexity level according to the specific characteristics of the learners in the specific class.
- The suggested types of questions sometimes overlap each other and cannot be separated completely from one another. This point is further addressed when relevant.
- A question can combine several types of questions in its different sub-paragraphs, as illustrated by an example given in the sub-section entitled "Combining several types of questions" in Section 5.
- Questions can literally be divided into two types: Pure algorithmic tasks and story-based algorithmic tasks (narrative). This perspective will be discussed in Section 5.

## 4. Types of Questions

*Type 1: Developing a Solution*

***Description***. A development question presents an open problem for which students are required to develop a solution in the form of a verbal algorithm, pseudo-code, or by using a specific programming language.

***Example***. Write a method that accepts an integer $n$ as input and returns the number of (integer) divisors of $n$.

***Variations***. A development question can address (a) a sequence of instructions; (b) a single method (as in the above example); (c) a whole program; or (d) a method with a specific efficiency (in the above example this can be $O(\sqrt{n})$ .

***Discussion***. This type of questions invites different solutions. In some cases, the differences are not meaningful; in others, the different solutions represent different algorithmic approaches. Variation (d) is not entirely an open question since a significant constraint must be met – the requested efficiency. In this case, students cannot just choose a solution to solve the problem and therefore, it is considered to be more difficult than the other variations and requires wider considerations.

*Type 2: Developing a Solution that Uses a Given Module*

***Description***. In this case, the development question relates to a predefined module. The student must present a solution to a given problem while considering and using a given module. Documentation of the module is included in the question and the student must use it in the developed solution.

***Example***. Write a method that accepts an integer value $n$ as input and returns the integer between 1 and $n$ with the largest number of divisors. Use the method *numberOfDividers(n)*, which accepts an integer value $n$ as input and returns the number of its divisors.

***Variations***. A development question that relates to a given module can be presented, among other ways, in one of the following forms: (a) write an instruction that invokes a given method; (b) write a method that uses a given method (like in the given example); (c) write a method that uses a given module a specified number of times; or (d) write a method that uses several different given methods.

***Discussion***. The fact that students must relate to a given module influences the development process of the solution. For example, in the case of a given method the student must match the developed method to a specific sub-task that the given method implements. This type of questions is considered more difficult than Type 1 questions because students must satisfy a constraint – the use of the given sub-task. The given module should not be a method but rather it can be, for example, a specified data structure (like Linked list) or class.

*Type 3: Tracing a Given Solution*

***Description***. A given code is presented and the students are asked to follow the code execution.

***Example pattern***. Present a tracing table that follows the execution of a given method. The table should include a column for each variable and for the code output.

***Variations***. A tracing question can involve following, for example on: (a) a complete program; (b) a single method; (c) a recursive method; or (d) the creation of objects. In addition, the following instructions can be used in each of the above variations: (1) follow the code execution with a given specific input; (2) follow the code execution with the student choosing the input; (3) follow the code execution with different specified inputs that are selected so as to guide the student to reveal what the given code does; (4) find different sets of inputs so that each set represents a different sequence by which the code is executed; or (5) find a set of inputs that yields a specific output.

***Discussion***. Variations 1–3 of the instructions can be considered to be closed questions. The student is required to trace a given code with a specified (given or self-chosen) input, and there is only one correct solution. Instructions (4) and (5) require the students to involve additional, deeper considerations and to examine the presented code more closely. It is not sufficient to understand different instructions; rather, it requires code analysis – what is the purpose of the code and how is it achieved. Clearly, more advance cognitive skills are needed in order to address these instructions in a meaningful manner.

*Type 4: Code Execution Analysis*

***Description***. A given code is presented and the student is asked to analyze some aspects of the code execution. This kind of questions is more complicated than tracing a given code since it requires the student to analyze code execution.

***Example pattern***. The following code includes a loop. Examine the code and answer the following questions:

(i) For what values of $x$ and $y$ will the loop not be executed at all?
(ii) For what values of $x$ and $y$ will the loop be executed exactly once?
(iii) For what values of $x$ and $y$ will the loop never end?

***Discussion***. This type of questions requires the student to understand the given code as a whole. Therefore, a higher level of thinking is needed in order to solve such questions than that needed to solve a tracing question. "Code execution analysis" questions relate mainly to two cognitive skills: (1) understanding programming structures; and (2) understanding the logic of a given code. Note that instructions (4) or (5) presented in the discussion of Type 3 questions – Tracing a given solution – can also be viewed as code execution analysis tasks.

*Type 5: Finding the Purpose of a Given Solution*

***Description***. A given code or algorithm to an unknown problem is presented and the student is asked to state the purpose of the solution – to determine what problem it solves.

***Example pattern***. Examine the given method and find the target of the method, that is, what is the problem that the method solves?

***Variations***. A "Finding the purpose of a given solution" question can relate to either: (a) a sequence of instructions; (b) a single method (like in the above example pattern); (c) a full program; or (d) a class of objects.

***Discussion***. Solving this type of questions requires a set of cognitive skills. In addition to an understanding of the code execution and the ability to trace it, a unique understanding and a unique skill, are required. These questions are considered harder than developing a solution for the same problem that the code solves. One reason for this is the need to comprehend someone else's way of thinking. To help students and guide them in solving this type of question, questions can contain scaffolding sub-questions. For example, a question can include several "Trace a given solution" sub-questions in the form of the instructions presented in the discussion of Type 3 questions, aimed at guiding the students to discover what the purpose of the code is.

*Type 6: Examining the Correctness of a Given Solution*

***Description***. A given problem and its solution are presented. The student is asked to determine whether the given solution is a correct solution to solve the given problem.

***Example***. The following method was written by a student as a solution for the following problem: Write a method that accepts an array of integers as input and returns *true* if all of the array's values are identical or *false* if they are not. Is the method correct?

```
public static boolean equalsValues (int[] arr) {
    for (int i = 0; i <arr.length; i = i + 2) {
        if (arr[i] != arr[i + 1])
            return false;
    }
  return true;
}
```

***Variations***. This type of question can be presented for different types of tasks: (a) state whether a given solution to a given problem is correct (as in the above example); (b) check whether a given solution is correct and explain your answer; (c) if the given solution is incorrect, give an example of an input that demonstrates its incorrectness; (d) if the given solution is incorrect, give an example of an input that will lead to a correct output, which could then lead to the conclusion that the given solution is correct; (e) if the given solution is incorrect, correct the solution by implementing the minimal required changes (without the "minimal" restriction, students may present a totally different solution); or (f) the given solution may contain more than one mistake, and the question may or may not state that explicitly.

***Discussion***. In order to solve this type of question, students should apply logic algorithmic thinking skills. Here, as in Type 5 questions, students must analyze a solution that may not correspond with their own way of thinking had they been asked to suggest a solution. However, since the purpose of the solution is given, these tasks are considered to be easier than Type 5 questions.

In the given example, the two minimal required corrections are: (i) change the increment of variable $i$ to 1 (instead of 2); and (ii) change the range of variable $i$ to $i <$ *arr.length* $-1$. Correction (i) is based on a logical consideration of the solution, while correction (ii) involves addressing the array index, which is a more technical consideration.

Additional variations of correctness questions may address syntactic mistakes. Such questions should be posed while introducing new instructions or structures. I do not recommend, however, using them at more advanced stages since they do not reflect an understanding of the algorithmic problem, and the compiler in fact directs the debugging of such mistakes.

*Type 7: Completing a Given Solution*

***Description***. A given problem and an incomplete solution of it are presented; some of the solution instructions are missing. The students are asked to complete the missing instructions so that the solution actually will solve the problem.

***Example***. The following method was written by a student as a solution for the following problem: Write a method that accepts an array of integers as input and returns the number of array elements that are greater than their two neighbors (the previous element and the subsequent element in the array).

```
public static int numberOfBiggers (int[] arr) {
    _____ ;
    for (int i =_____ ; i<_____ ; i++) {
        if ( _____ )
            _____ ;
    }
    return _____ ;
}
```

***Variations***. A "Completing a given solution" question can vary in the extent to which instructions are missing. The number of missing instructions should be decided on, taking into consideration the effect on the question's difficulty and complexity. For example, if the objective is to focus on the use of a Boolean flag, the missing instructions should only be those that relate to the flag; or, if the loop limits are the target, the limits should be missing and perhaps the increase in the loop control variable as well. The extent of missing instructions is quite significant in the above example.

***Discussion***. This type of question also requires students to understand the logic of the given solution. Question difficulty is determined according to the students' level and stage of learning although for each subject there is a relatively simple completion of a given solution for which the logic of the solution is straightforward and not so difficult to understand. Still, other, more challenging questions exist and instructors should be aware of this potential complexity. For example, asking students to complete instructions for a given bubble sort algorithm with missing meaningful instructions without introducing the rational of this sorting approach, is considered a difficult question.

In general, the missing instructions can relate to one or more aspects of the algorithm and the instructor should consider whether to focus on one or more aspects. In the above example, the missing instructions involve three aspects of the algorithm: the counter control (initializing, increasing, and returning); the range of the loop (the first and the last array elements should not be accessed in the loop because they do not have two neighbors); and the specific condition to be checked.

*Type 8: Instruction Manipulations*

***Description***. A problem and its solution are given. Students are asked to address different manipulations performed on the solution.

***Example***. The following method is a version of a selection sort solution.

```
public static void selectionSort (int[] arr) {
      int p, temp;
(1)   for (int i = 0; i < arr.length −1; i + +) {
(2)         p = i;
(3)         for (int j = i + 1; j <arr.length; j + +) {
                  if (arr[j] < arr[p])
                        p = j;
            }
(4)         if (p! = i) {
                  temp = arr[i];
                  arr[i] = arr[p];
                  arr[p] = temp;
            }
      }
      }
```

Answer the following questions and explain your answers:

(i) Will the algorithm correctness be affected if the instruction in line (2) is removed and the instruction in line (3) is replaced by the following instruction:

```
(3)      for (int j = i; j <arr.length; j + +) {
```

(ii) Will the algorithm correctness be affected if the instruction in line (4) is removed and the contents of the two array elements are exchanged anyway?

(iii) Will the algorithm correctness be affected if the entire body of the loop (1) (lines 2–4) is replaced by the following instructions?

```
for (int j = i + 1; j <arr.length; j + +) {
      if (arr[j] < arr[i]) {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
      }
}
```

***Variations***. A "Manipulation of instructions" question can involve: (a) adding instructions; (b) removing instructions; (c) changing instructions; or (d) replacing instructions. The question can address the target of a specific code or the tracing of the changed code, or it can examine differences between outputs.

***Discussion***. Questions that manipulate an existing solution enable students to focus on the more meaningful aspects of the algorithm. Course instructors can then lead a discussion of such manipulations in order to clarify the essence of a given solution, as well as other computer science topics. For instance, a discussion about a change made to instructions can also highlight the concept of generalization. Students can be instructed to make a slight change to a given method so as to generalize the original method and solve a broader task. For example, a method that sorts array can be changed slightly so that it

sorts a section of the array between two given indexes. After the change, the method is more general and can sort different sections of an array, as well as the entire array (with the indexes 0 and the array length $-1$).

*Type 9: Efficiency Estimation*

***Description***. Students are required to estimate the efficiency of a given solution.

***Example pattern***. Estimate the efficiency of a given method in terms of Big-O notation. Explain your analysis.

***Variations***. This type of questions can be presented on different levels of cognitive complexity: (a) an example pattern that enables an early discussion on efficiency is: Focus on the loop in a given method, how many times is it executed? The following variations are more complicated: (b) estimate the efficiency of a specific method (as in the example pattern); (c) estimate the efficiency of a method that invokes another method, taking into account the efficiency of the invoked method; (d) compare the efficiency of different methods that solves the same task; (e) estimate the efficiency of a recursive method; or (f) develop a solution with a required specific efficiency.

***Discussion***. Questions that require students to relate to efficiency can be integrated quite early on in the teaching and learning processes. Instructors should not wait till they teach complicated algorithms in order to teach the concept of efficiency. Questions, such as that presented in variation (a), demonstrates the basic idea of the efficiency concept, which, in general, is considered to be abstract and difficult to understand.

Note that variation (f) is actually a development question (Type 1) with an efficiency restriction that requires a combination of cognitive skills. Students should not be satisfied with finding an algorithmic idea that solves the given problem; rather, they should estimate its efficiency and if it does not satisfy the restriction mentioned in the question, they should seek a different solution.

*Type 10: Question Design*

***Description***. Students are asked to design a question on their own.

***Example 1***. Design a question that checks the understanding of the sort-merge algorithm.

Students' answers to this question (which are questions) can be based, for example, on tracing regular and/or extreme cases.

***Example 2***. Design a question whose solution requires the use of a method that finds the most frequent value in an array.

An example of such a question is: Write a method that for each of a school's 10th grade classes, accepts student grades on a computer science test as input and returns the frequent grade in each class.

***Variations***. A design question can relate, among other things, to: (a) clauses in a given question. Students are asked to compose additional clauses for a given question that, in their opinion, help clarify some extreme case; (b) a question whose solution requires the use of a given method (see above Example 2); (c) a question that checks the understanding

of a specific concept or algorithmic idea (see above Example 1); or (d) an entire test or worksheet that examines a specific learning unit.

***Discussion***. This type of question changes the learners' point of view. In addition to the experience gained by examining a question from an instructor's point of view, it encourages the students to think about the learning concepts. It leads learners to reflect on what was learnt, the main involved concepts, the sub-contents of the concepts, and furthermore, to think about checking their own understanding. In addition, the design of questions is a kind of active learning that encourages creativity.

*Type 11: Programming Style Questions*

***Description***. Students are asked to examine the programming style of different solutions proposed for the same task.

***Example pattern***. Look at the different, correct solutions for a given problem. Examine the solutions and state, in your opinion, which is the best solution. Explain your choice.

***Variations***. The different solutions given for the problem should differ in one aspect only, according to the instructor's decision, such as: (a) different kinds of loops; (b) the need to use an array for the solution; or (c) different algorithmic approaches (for example: giving two correct solutions to a problem and asking the students which of them is "nicer" and why). If the instructor decides to integrate several aspects into the question, the different aspects can be presented explicitly in the question and the students can be asked to address the solutions according to each aspect. Alternatively, the students can be asked, first to address the different aspects and then discuss the solutions themselves.

***Discussion***. This type of question enables to discuss various aspects of programming styles, while comparing different solutions. The different aspects can be for example: modularity, complexity, programming style, readability, or memory uses. Such a discussion can increase the abstraction level of student thinking.

*Type 12: Transforming a Solution from One Representation to Another*

***Description***. A problem and its solution are presented to the students in a specific syntax or paradigm, and the students must transform the solution into a different syntax or paradigm.

***Example pattern 1***. The given loop is implemented using a *while* loop structure. Transform the loop so that it is implemented by a *for* loop structure.

***Example pattern 2***. The given method is implemented using a *while* loop structure. Transform the method into a recursive method that achieves the same target.

***Example pattern 3***. The following method sorts an array of integers according to the imperative approach implemented in Java. Transform the method so as to reflect the functional approach and implement it in Scheme.

***Variations***. The different representations can be: (a) between programming paradigms (as in Example pattern 3); (b) within the same programming paradigm but between programming languages; (c) within the same programming language but between structures

(as in Example pattern 2 or, for example, the transformation from a nested if statement to a switch-case statement); or (d) within the same programming language but between different algorithmic approaches (as in Example pattern 2).

***Discussion***. The focus of this kind of question should be placed on qualitative aspects rather than syntactical aspects, where qualitative aspects mean, for example, problem analysis according to two different programming paradigms or the transformation of an imperative solution into a recursion solution in the same programming language. Such tasks require abstract thinking processes.

Similar to the "programming style questions" (Type 11), qualitative transformation questions enable to concentrate on core computer science concepts. Such questions lead students to explore different ways of thinking in problem-solving situations. Since this kind of question demands a high level of abstraction, it is not necessary suitable for all students. Obviously, transformations between two programming paradigms can be carried out only after the two said programming paradigms have been learned.

It is my opinion that transformations that involve only syntactic issues, for example, from pseudo-code to any formal language, do not involve problem-solving skills. Such tasks may be required in order to practice ways of writing, but they do not involve meaningful CS concepts.

## 5. Global Aspects of Question Types

In this section, I present three additional approaches to questions. The first involves combining several types of questions, the second refers to story questions, and the third to closed questions. The three approaches can be applied to most of the twelve question types presented above.

*Combining Several Types of Questions*

Despite the attempt to uniquely classify computer science questions in the above list of question categories, in most cases questions are a combination of several types, as the following example illustrates (while others may not be classifiable by the presented collection at all).

***Example***. The target of Methods A and B presented in Fig. 1 is to determine whether or not an integer n is a prime number.

Following is a list of questions that can be asked separately or in any combination according to the pedagogical purposes of the instructor.

(i) Check the correctness of the solutions. Do they solve the problem?
*Type6: correctness.*

(ii) What is the purpose of each method? (in case the problem is not indicated).
*Type5: find the purpose.*

(iii) Trace each method given $n = 19$.
*Type3: trace.*

```
//Version A
    public static boolean prime (int n) {
        for (int i = 2; i < n; i++) {
            if (n%i == 0)
                return false;
        }
        return true;
    }
-------------------------------------------
// Version B
    public static boolean prime (int n) {
        if (n%2 == 0)
            return false;
        for (int i = 3; i < n; i = i + 2) {
            if (n % i == 0)
                return false;
        }
        return true;
    }
```

Fig. 1. Combining several types of questions.

(iv) For each method, determine how many times the loop is executed for $n = 19$.
*Type4: code execution analysis.*

(v) Find a value of n for which the loop in Version B is executed 10 times. Is there only one answer?
*Type4: code execution analysis.*

(vi) What is the efficiency of each of the two methods?
*Type9: efficiency.*

(vii) Is the solution still correct if you change the upper loop limit in Version B to n/2 instead of n? If it is, what is the method efficiency after the change?
*Type8: manipulation; Type6: correctness; Type9: efficiency.*

(viii) Is the solution still correct if you change the loop limit in Version B to $\sqrt{n}$ instead of $n$? If it is, what is the method efficiency after the change?
*Type8: manipulation; Type6: correctness; Type9: efficiency.*

*Story Questions*

Questions can literally be divided into two types: pure-algorithmic tasks and narrative-algorithmic tasks. ***Pure-algorithmic*** tasks are problems that directly and explicitly address program structures and program variables, and present the task in that context. ***Narrative-algorithmic*** tasks are problems that do not directly address either the required program structures or the required program variables; the problem to be solved is embedded in a story and in order to solve it, learners must recognize both what is given and

Table 1
Tasks presented as pure-questions and as narrative-questions

| Task | Pure question | Narrative question |
|---|---|---|
| Find the maximum of a list of numbers. | Write a method that accepts a list of integers as input and returns the maximum value of the list. | During a sports day, each of the 30 students in 5 classes participated in two competitions, the high jump and the long jump. Write a program that inputs, for each class, each student's two results, and outputs the best high jump result and the best long jump result for each class. |
| Check whether a given array is sorted. | Write a Boolean method that accepts an array as a parameter and returns a Boolean value if the array is up-sorted. | A teacher wishes to encourage her or his students, and so gives them special certificates if their test scores improve. Write a method that accepts the list of each student's grades as input and determines whether he or she deserves a certificate of recognition. |
| Exchange characters with their successive characters according to the Unicode table. | Write a method that accepts an array of characters as a parameter and changes the array so each character is replaced by its successive character according to the Unicode table. | A message that is to be sent between financial partners must be encoded. The message includes words, spaces, and periods. Write a method that accepts a string with a message as a parameter, and returns a coded message in which each letter is replaced by its successive letter according to the ABC. The letter Z is to be replaced by the letter A. Spaces and periods remain unchanged. |

what the target of the problem is. Specifically, learners should decide which elements are relevant to solving the problem and which are irrelevant. Most of the examples presented in the list of question types are pure examples in which the task is presented directly. Table 1 presents several tasks as both pure-questions and as narrative-questions.

It is important that CS educators are aware of the differences between the problem-solving skills required to solve pure-questions as opposed to narrative-questions. A pure-algorithm question directs the learner to the core of the task; in narrative question, on the other hand, students must reveal the task and determine what the specific assignments are. Since in the real world, most problems are based on narratives, solving this type of questions is an important skill that computer science students should acquire. Still, these questions are usually more complicated.

When teaching new computer science content, I recommend that several stages be followed in which questions of the two types are addressed. First, present a story that embeds the new learned topic so that the class grasps the essence and target of the new topic, which will enhance their programming tool arsenal. Second, focus for a while on pure-questions to enable a gradual knowledge construction process of the new tool or structure in the context of programming. Finally, integrate narrative questions into the subsequent stages of the teaching process.

*Closed Questions*

The common concept of ***closed questions*** refers to questions that present a list of possible answers of which the learner is expected to choose and mark one. The most frequently encountered types of closed questions are multiple choice questions and true/false questions. In fact, what really is ”closed” are the answers, not the questions. The twelve question types presented above could be discussed in terms of questions: (a) that can be presented naturally only as open questions, where learners must present their own answers; or (b) that can be presented naturally as either open questions or closed questions, where the learners must choose and mark an answer from a set of given answers.

In what follows, the 12 types of questions is divided into groups with relation to the option to be presented as closed questions or not.

- *Types of questions that can be presented as closed questions*
  The types of questions that can be presented naturally as closed questions are: Type3 – Tracing a given solution; Type4 – Code execution analysis; Type5 – Finding the purpose of a given solution; Type6 – Examining the correctness of a given solution; and Type9 – Efficiency estimation.
  ***Example***. A closed question of Type6 can give a list of methods that aim to solve the same task. The learner is asked to indicate for each such method whether or not it is correct.
- *Types of questions that can not be presented as closed questions*
  The types of questions that can not be presented naturally as closed questions are: Type1 – Developing a solution; Type2 – Developing a solution that uses a given module; Type10 – Question design; Type12 – Transforming a solution from one representation to another.
  These types of questions obviously require that the learner develops a solution that satisfies the instructions.
- *Types of questions that can not be presented naturally as closed questions*
  The remaining types of questions can be presented as closed questions but this is not their natural form: Type7 – Completing a given solution; Type8 – Instruction manipulations; Type11 – Programming style questions.
  ***Example***. A closed question of Type7 can give a list of optional instructions to be added to a given code that solves a given task in a specific place. Students are asked to indicate which of them is suitable to be added.

## 6. Applying the Different Question Types to Different CS Contents

As stated above, the question types presented in this paper relate to programming-like questions; still, most of these types can be implemented on other CS contents as well. Table 2 displays specific variations of the twelve question types as could be implemented, for instance, in the field of Automata Theory.

Table 2

Question types and representative examples in automata theory

| **Type of question** | **Example pattern** |
|---|---|
| Type 1: Developing a solution | Design a finite automaton $A$ that recognizes regular language $L$. |
| Type 2: Developing a solution that uses a given module | Given finite automaton $A_1$ that recognizes language $L_1$ and finite automaton $A_2$ that recognizes language $L_2$, design a finite automaton that recognizes the language $L_1 \cup L_2$. |
| Type 3: Tracing a given solution | Given a pushdown automaton $P$ and the word $w$, show the sequence of states that $P$ goes through when processing $w$. |
| Type 4: Code execution analysis | Given a finite automaton $A$, find:<br>– a word whose processing will terminate in an acceptable (final) state;<br>– a word whose processing will terminate in an unacceptable (not final) state;<br>– a word whose processing will terminate in the trap state. |
| Type 5: Finding the purpose of a given solution | Given a Turing machine *T,* determine what language it processes. |
| Type 6: Examining the correctness of a given solution | Does Turing machine $T$ recognize language $L$? |
| Type 7: Completing a given solution | Complete the pushdown automaton $P$ so it recognizes language $L$. |
| Type 8: Manipulating instructions | Given a Turing machine $T$, what language does the machine recognize if the path from state $q_1$ to state $q_2$ is replaced by the following given path? |
| Type 9: Estimating "efficiency" | Given a finite automaton $A$ that recognizes language $L$, find a different finite automaton that recognizes the same language with fewer states. |
| Type 10: Designing a question | Design a question that requires the presentation of a BNF grammar for an irregular language. |
| Type 11: "Programming" style questions | Given three different pushdown automatons that recognize language $L$, examine the automatons and state which of them, in your opinion, is more "qualified". |
| Type 12: Transforming a solution from one representation to another | Given a Turing machine $T$, present a BNF grammar that expands the same language. |

## 7. Summary

This paper explores and discusses variety of types of questions that can be used by computer science educators in different teaching situations. The paper illuminates the important role computer science educators have in introducing their students to different types of questions that can be used alongside learning-teaching processes. The exploration of different types of questions and their variations deepens students' understanding of the learnt computer science concepts and help them acquire a variety of cognitive skills. It also provides different learners with the opportunity to express their knowledge, which is

elicited by different sorts of questions, and to refine their understanding of complex concepts. Furthermore, the use of a variety of question types provides intellectual challenges and maintains learners' concentration, interest, and motivation.

Further work can be done to analyze the types of questions identified according to known taxonomies.

**Acknowledgments.** Many thanks to Prof. Orit Hazzan and Dr. Tami Lapidot, my colleagues at the Technion – Israel Institute of Technology, for their helpful review and for encouraging me to publish this work.

## References

Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, J., WITTROCK, M.C. (Eds.) (2001). *A Taxonomy for Learning and Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman.

Batory, D., Sarvela, J.N., Rauschmayer, A. (2004). Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6), 355–371.

Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R. (1956). *Taxonomy of Educational Objectives Handbook 1: Cognitive Domain*. London, Longman Group Ltd.

Brand-Gruwel, S., Wopereis, I., Vermetten, Y. (2005). Information problem solving by experts and novices: analysis of a complex cognitive skill. *Computers in Human Behavior*, 21(3), 487–508.

Byckling, P., Sajaniemi, J. (2006). A role-based analysis model for the evaluation of novices' programming knowledge development. In: *Proceedings of the Second International Workshop on Computing Education Research (ICER'06)*, Canterbury, United Kingdom, 85–96.

Chamillard, A.T., Braun, K.A. (2000). Evaluating programming ability in an introductory computer science course. In: *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education (SIGCSE '00)*, Austin, Texas, United States, 212–216.

English, J., Rosenthal, T. (2009). Evaluating students' programs using automated assessment: a case study. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*, Paris, France, 371–371.

Ginat, D. (2003). The novice programmers' syndrome of design-by-keyword. In: *Proceedings of the 8th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '03)*, Thessaloniki, Greece, 154–157.

Ginat, D. (2009). Interleaved pattern composition and scaffolded learning. In: *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*, Paris, France, 109–113.

Jones, K.O., Harland, J., Reid, J.M., Bartlett, R. (2009). Relationship between examination questions and Bloom's taxonomy. In: *Proceedings of the 39th IEEE International Conference on Frontiers in Education Conference*, San Antonio, Texas, USA, IEEE Press, Piscataway, NJ, 1314–1319.

Muller, O., Ginat, D., Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. *ACM SIGCSE Bulletin*, 39(3), 151–155.

Pomerantz, J. (2002). Question types in digital reference: an evaluation of question taxonomies. In: *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '02)*, Portland, Oregon, USA, ACM, New York, NY, 404–404.

Sajaniemi, J. (2005). Roles of variables and learning to program. In: Jimoyiannis, A. (Ed.), *Proceedings of the 3rd Panhellenic Conference "Didactics of Informatics"*, , University of Peloponnese, Korinthos, Greece. Available at: `http://cs.joensuu.fi/~saja/var_roles/abstracts/didinf05.pdf` [Last access at March 29, 2012].

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J.K., Padua-Perez, N., (2006). Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In: *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE '06)*, Bologna, Italy, ACM, New York, NY, 13–17.

Sorva, J., Karavirta V., Korhonen, A. (2007). Roles of variables in teaching. *Journal of Information Technology Education*, 6, 407–423.

Thompson, E., Luxton-Reilly, A., Whalley, J., HU, M., Robbins, P., (2008). Bloom's taxonomy for CS assessment. In: *Proc. Tenth Australasian Computing Education Conference (ACE 2008)*, Wollongong, NSW, Australia. CRPIT, 78. In: Simon and Hamilton, M. (Eds.), ACS, 155–162.

Vasconcelos, J. (2007). Basic strategy for algorithmic problem solving. Retrieved from: `http://www.cs.jhu.edu/~jorgev/cs106/ProblemSolving.html` [Last access at June 2, 2010].

Venables, A., Tan, G., Lister, R., (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. In: *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*, Berkeley, CA, USA, ACM, New York, NY, 117–128.

Yng, A., Wu, J., Wang, L. (2008). Research and design of test question database management system based on the three-tier structure. *WTOS*, 7(12), 1473–1483.

**N. Ragonis** is the chair of the Curriculum Committee and academic advisor, School of Education, Beit Berl College. She is a lecturer in the Department of Computer Science, and served for ten years as the head of the Department. Teaches courses related to computer science (e.g., OOP, graph theory, computational models), to the didactics of computer science and to information technologies such teaching and learning in online environments and query learning with spreadsheets. She is also adjacent senior lecturer, Department of Education in Technology and Science, Technion – Israel Institute of Technology. Her activities in the past twenty years concerns: educational research mainly focused on cognitive aspects of teaching and learning of computer science and integrating tutoring activities in teachers education; in-service teachers training; development of high school text books as well as teachers guides; and serve as a member of the management staff of "Machsava" (Thought), the Israeli National Center for High School Computer Science Teachers, Technion – Israel Institute of Technology and Weizmann Institute of Science.