Space Complexity in CS1

Ela Zur[1], Tamar Vilner[2], Judith Gal-Ezer[3]

[1]The Open University of Israel, 108 Ravutzki St., Raanana, Israel, ela@openu.ac.il

[2]The Open University of Israel, 108 Ravutzki St., Raanana, Israel, tami@openu.ac.il

[3]The Open University of Israel, 108 Ravutzki St., Raanana, Israel, galezer@openu.ac.il

The importance of introducing efficiency of algorithms in the early stages of an undergraduate study program in Computer Science is widely acknowledged, as are the difficulties encountered when introducing the concept. Whenever efficiency is introduced, the focus is on time efficiency, while space efficiency is only briefly mentioned. The difficulties that students have with regard to space complexity motivated our study, which we conducted in two consecutive stages. In the first part of the research, we found misunderstandings relating to space complexity. Relying on these findings, we changed our way of teaching space complexity, and later conducted the second stage of the research to examine the implications of the change.

Keywords

CS1, Space Complexity, Misconceptions

1. Background

Algorithms are the spirit of computing, and good algorithm design is crucial to the performance of all software systems, as is the ability to select algorithms appropriate for specific purposes and to recognize the possibility that sometimes no suitable algorithm exists. The study of algorithms gives the student insight into the problems involved in providing techniques for solutions that are independent of programming languages, or other aspects of implementation. The design of efficient algorithms to solve algorithmic problems is one of the important areas of research within computer science (CS) and therefore central to computer science education [4, 5, 6, 7].

Computing Curricula 2001 [8] notes that a large part of the core and elective course material is devoted to algorithms. Efficiency and complexity are pervasive themes throughout the study of algorithms. Algorithm complexity is measured in terms of time and space. Time complexity is measured by the number of elementary actions carried out during the execution of the algorithm, while space complexity is measured by elements such as the number and size of the data structures used. In a previous study, we described difficulties we encountered while teaching time complexity in CS1 at the Open University of Israel (OUI), and the approach we suggested to help the students [2]. In the current paper, we will relate to teaching space complexity.

The Open University of Israel is a distance learning university, which offers a variety of undergraduate and graduate programs [13]. The OUI is similar to other universities in its pursuit of excellence and its commitment to superior scientific and scholastic standards. However, it differs in that it is open to all those who wish to study a single course or a number of courses, or to pursue a full program of study towards a Bachelor's degree. Undergraduate enrolment does not require matriculation or any other certificate from an educational institution; students' academic achievements are the key to their success.

At the OUI, CS1 is similar to introductory courses given in other universities, and includes the topics recommended in *Computing Curricula 2001*: basic logic, algorithms and problem solving, fundamental data structures, fundamental programming constructs, recursion, fundamental computing algorithms, basic computability, etc. [8]. We introduce efficiency relatively early, which encourages students to consider alternative designs of algorithms, to analyze various algorithms, and to formulate them correctly [5, 6].

However, such early introduction may lead to difficulties: the problems discussed at early stages of the introductory course are almost always 'toy' problems, making it difficult to convince students that a more efficient algorithm is indeed needed. Also, the analysis of algorithm efficiency requires mathematics with which students are not always familiar when they take the introductory course.

Misconceptions were encountered [3, 4]; for instance, students often bring up the myth of the speed and the growing capacity of the computer, saying that computers are so incredibly fast that there is no real problem. This belief is, of course, groundless: time and space are crucial in almost every use of the computer. Many examples can be found to show that whatever the speed of computers now or in the future, speeding up the execution of algorithms will always be important (see, for example, [7]). Moreover, an algorithm might simply be too expensive and thus unacceptable.

As is common in CS1 courses in other universities and in many standard textbooks, we focus on the time aspect. We first present algorithmic problems with unreasonable algorithmic solutions, because we believe that this will increase motivation to learn the topic. Students begin to understand how important it is to be able to analyze the complexity of an algorithm and to realize that some algorithms are unreasonable even if we have a very fast and big computer. We then explain how to measure complexity, and how to compare the time complexity of different algorithms. We explain how critical it is to reduce the running time of algorithms by an order of magnitude and not only by a constant factor. As to space complexity, we introduce the concept and discuss it using different examples.

Years of experience with the introductory course showed that students have difficulty perceiving time complexity. After we felt that we helped students to cope with time complexity by introducing a new approach [2], we turned to examining the perception of space complexity. The professional literature devoted to CS education discusses misconceptions relating to fundamental concepts, mainly in programming [1,9,10,11,12], but we did not find studies that discuss space complexity. We therefore conducted a study to examine how CS1 students perceive space complexity.

Our study consists of two stages. The first stage was carried out on 247 CS1 students during the fall semester of 2004. After analyzing the results, we changed our teaching approach, and conducted the second stage of the study, which included 202 CS1 students in fall 2006.

2. The Study - Part One

2.1 Research Questions

We posed two main research questions:

1. To what extent are students successful in analyzing the space complexity of a given program?
2. What are the difficulties in understanding the concept of space complexity?

2.2 Research Population

Our study was carried out on 247 CS1 students during the fall semester of 2004. Only 116 students (47%) passed the final examination, probably due to the university's open admissions policy.

2.3 Research Instruments

The students could take one of two final exam sittings, each of which included a question relating to space complexity, as detailed below:

Question 1a (First sitting)

Suppose *mat* is a given square matrix (2 dimensional array) of length $n \times n$ (n is a constant). Each element of *mat* is an integer.

We define the value ***Big-of-Small***s as the largest of the numbers which are the smallest numbers in their row.

a. Write an efficient function which gets a square matrix *mat* filled with integers, as a parameter, and returns the Big-of-Smalls value in the matrix *mat*.

b. What is the time complexity and what is the space complexity of the function you wrote?

Example:

In this matrix (n = 4)

-2	6	**-5**	1
2	**-8**	-4	18
5	77	7	**3**
33	3	**-1**	0

The value Big-of-Smalls is 3, because it is the biggest among -5, -8, 3 and -1.

Question 1b (Second sitting)

Suppose *mat* is a given square matrix (2 dimensional array) of length $n \times n$ (n is a constant). Each element of *mat* is an integer.

We define the matrix to be a **Horse-Matrix** if one of its squares has the following attribute: taking a knight's step (like in chess) from a square lands on a square whose value equals that of the current square.

Note that in the matrix *mat*, the square mat[i][j] has 8 possibilities of knight's step squares:

mat [i ± 1] [j ± 2], *mat* [i ± 2] [j ± 1] (if the indices are legal, i.e., they are bigger than or equal to 0 and smaller than n).

a. Write a function which gets a square matrix *mat* filled with integers, as a parameter, and returns 1 if the matrix is a **Horse-Matrix** and 0 otherwise.

b. What is the time complexity and what is the space complexity of the function you wrote?

Example:

This is a **Horse-Matrix** (n = 5)

5	1	1	0	**5**
1	1	**5**	0	7
5	1	3	1	**5**
0	**5**	1	**5**	3
1	2	0	1	1

Please notice that the gray square *mat*[1][2] has 6 other squares in a knight's step from it, and the value in all of them is the same as that of the gray square: 5.

2.4 Findings

119 students took the first sitting of the exam. Of these, 57 passed.

For part a (Question 1a), students gave two solutions:

- The best solution is to find the smallest number in each row, while comparing each of these smallest numbers to the current highest value. The code for the best solution is described in Program 1.
  77% of the students who passed the exam (44 out of 57) gave this solution.
- The second solution, which was less efficient in its space aspect, was to find the smallest number in each row, and to place it in an additional one-dimensional array, then to pass over this array to find the highest value in it. The code for this solution is described in Program 2.
  23% of the students who passed the exam (13 students) gave this solution.

In part b, we asked the students to write the time and space complexity of their solutions. We found that 56 of 57 students (98%) answered correctly on time complexity - $O(n^2)$. We can see that the students internalized the fact that when they scan over a 2-dimensional array, the time complexity is $O(n^2)$.

```
int bigOfSmalls (int mat[n][n])
{
  int max = mat[0][0];
  for (int i=0; i<n; i++)
  {
        int min = mat[i][0];
        for (int j=1; j<n; j++)
              if (mat[i][j] < min)
                    min = mat[i][j];
        if (min > max)
              max = min;
  }
  return max;
}
```

Program 1 The code for the best solution to question 1a.

```
int bigOfSmallsWithArray (int mat[n][n])
{
  int minArray [n];

  for (int i=0; i<n; i++)
  {
        int min = mat[i][0];
        for (int j=1; j<n; j++)
              if (mat[i][j] < min)
                    min = mat[i][j];
        minArray[i] = min;
  }

  int max = minArray[0];

  for (i=1; i<n; i++)
        if (minArray[i] > max)
              max = minArray[i];

  return max;
}
```

Program 2 The code for the solution to question 1a, which uses an additional array

Pleased with these results, we were disappointed to find a different result on the space complexity question. 19 of the 44 students who gave the best solution, wrote the correct answer: that the space complexity is constant - *O*(1). Of the 13 students who wrote the solution using the additional array, only 6 gave the correct answer regarding space complexity for their solution - *O*(n).

The other 32 students were divided: 16 students did not answer on space complexity at all, and the other 16 gave an interesting answer - $O(n^2)$. It seems they counted the matrix which holds the numbers.

Figure 1 describes the distribution of the answers to the space complexity question.

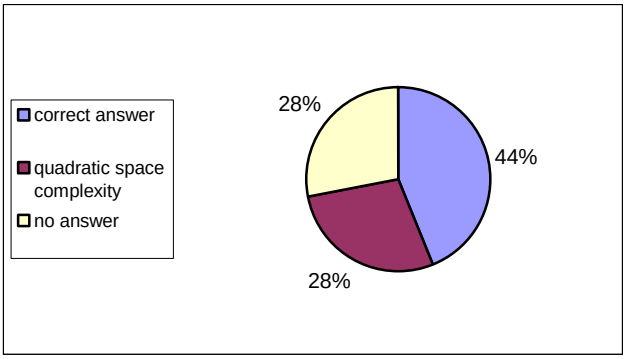

Figure 1 Distribution of the answers to space complexity in question 1a.

The second sitting of the exam was taken by 128 students, 59 of whom passed.

Program 3 describes one of the correct solutions for part a.

```
int inRange (int row, int col)
{
   return (row >= 0) && (row < n) &&
          (col >= 0) && (col < n);
}
int horseMatrix (int mat[n][n])
{
  for (int i=0; i<n; i++)
   for (int j=0; j<n; j++)
   {
     if (inRange(i-2, j-1))
      if (mat[i-2][j-1] != mat[i][j])
            continue;
     if (inRange(i-2, j+1))
      if (mat[i-2][j+1] != mat[i][j])
            continue;
     if (inRange(i-1, j-2))
      if (mat[i-1][j-2] != mat[i][j])
            continue;
     if (inRange(i-1, j+2))
      if (mat[i-1][j+2] != mat[i][j])
            continue;
     if (inRange(i+1, j-2))
      if (mat[i+1][j-2] != mat[i][j])
            continue;
     if (inRange(i+1, j+2))
      if (mat[i+1][j+2] != mat[i][j])
            continue;
     if (inRange(i+2, j-1))
      if (mat[i+2][j-1] != mat[i][j])
            continue;
     if (inRange(i+2, j+1))
      if (mat[i+2][j+1] != mat[i][j])
            continue;
     return 1;
   }
 return 0;
}
```

Program 3 The code for one of the correct solutions to question 1b.

90% of the students who passed the exam (53 students) gave this solution.

In section b, we found that 51 of the 59 students (86%) answered correctly on time complexity - $O(n^2)$. As to space complexity, here we found that 17 of the 59 students gave the right answer, saying that the space complexity was constant - $O(1)$. 17 students included the matrix in their space complexity and thus their answer was $O(n^2)$. 25 students did not answer the question.

Figure 2 describes the distribution of the answers to the space complexity.

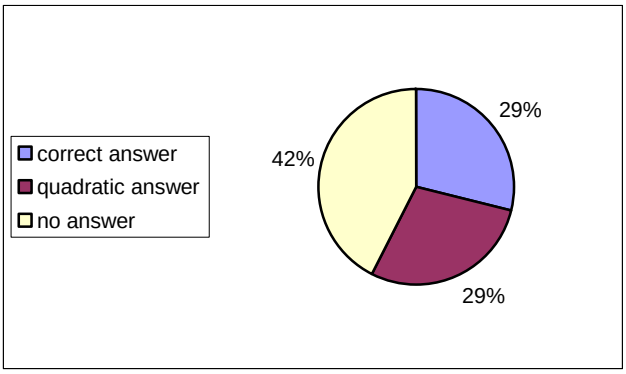

Figure 2 Distribution of the answers to space complexity in question 1b.

2.5 Discussion

We were disappointed to find that so many students have difficulty understanding the nature of space complexity. We realized that the students find it hard to distinguish between the amount of memory that is inherent to the problem (i.e., the amount of the space of the data structure given in the problem) and the additional space needed to solve it. These students thought that in computing space complexity, one has to add the space of the data structures that are inherent to the problem.

In the first exam sitting, 28% of the students who passed the exam (16 out of 57), said that the space complexity was $O(n^2)$, since they computed the matrix size as well. The same happened in the second exam sitting: 29% of the students measured the matrix as a part of the algorithm's space complexity, though it is a part of the problem and not of the algorithm.

3. The Study - Part Two

Following these results, we decided to place more emphasis on teaching the concept of space complexity. One of the examples we used, which we thought would help them understand the difference between the computer memory used in the algorithm itself and the memory inherent to the problem, was the binary search algorithm: When measuring the time of the binary search algorithm, one does not add the time needed for inserting the data into the array (which takes linear time) or for the sorting of the array (at least $O(n \log_2 n)$). The only actions counted are those performed by the binary search algorithm. That is why the time complexity of the binary search algorithm is logarithmic ($O(\log_2 n)$). All the other actions (like inserting the data and sorting the array) are already given and thus are inherent to the problem. The same happens in calculating the space complexity. The algorithm itself uses

only a few variables and its space complexity is fixed (*O*(1)). One must not add the array itself (which is linear (*O*(n)) to this calculation.

We expected that if the students had no difficulty understanding that the time complexity of the binary search algorithm is logarithmic, and thus did not add the time for inserting the data into the array or for sorting the array, they should understand that when calculating the space complexity, they compute only the computer memory used by the algorithm. Unfortunately, the results of the first stage of our study showed us that this is not the case. It turned out that the students did not transfer from measuring time complexity to measuring space complexity. We therefore realized that we could not expect students to make this transfer by themselves, and that we have to elaborate on this example and explicitly compare it to space complexity. Thus, we changed our teaching approach and after implementing it, conducted the second part of the study to find out whether there was any improvement.

3.1 Research Questions

To what extent are students successful in analyzing space complexity, after focusing on the concept in class?

3.2 Research Population

Our study was carried out on 202 CS1 students during the fall semester of 2006.

3.3 Research Instruments

The students could take one of two final exam sittings, each of which included a question relating to space complexity. Due to space limitations, we will only present the relevant question from the first exam sitting, question 2, below.

Question 2

Suppose *arr1* and *arr2* are given arrays of length n (n is a constant). Each element of *arr1* and of *arr2* is an integer.

Consider the following method:

```
boolean what (int arr1[], int arr2[])
{
    for (int i= 0; i<n; i++)
       for (int j=0; j<n; j++)
            if (arr2[j] < arr1[i])
                return false;
    return true;
}
```

a. What task does the function perform? Explain briefly what the function does in general terms, not how it executes the task.
b. What are the time complexity and the space complexity of the function above?
c. Write a function which performs the same task but which is an order-of-magnitude (not a constant factor) improvement in time complexity. A function with greater (time or space) complexity will not get full credit.
d. What are the time complexity and the space complexity of the function you wrote?

3.4 Findings

112 students participated in the first exam sitting.

The task performed by the function (part a of the question) was to find out whether all of the elements of *arr2* are greater than or equal to all the elements of *arr2*.

Part b - the time complexity is $O(n^2)$, and the space complexity is $O(1)$. 99% of the students (109 out of 112) answered correctly on the time complexity question, and 75% (83 out of 112) answered correctly on the space complexity.

For part c, students gave two solutions:

- The best solution is to find the smallest number in *arr2*, and to compare it with the biggest number in *arr1*. The time complexity of this solution is $O(n)$. The code for the best solution is described in Program 4.

  55% of the students gave this solution.
- The second solution, which was less efficient in its time aspect, was to sort the two arrays and then to compare the last element in *arr1* with the first element in *arr2*. The time complexity of this solution is $O(n \log_2 n)$.

  31% of the students gave this solution.

```
boolean what (int arr1[], int arr2[])
{
    int maxArr1 = arr1[0];
    int minArr2 = arr2[0];
    for (int i= 1; i<n; i++)
    {
      if (arr1[i] > maxArr1)
          maxArr1 = arr1[i];
      if (arr2[i] < minArr2)
          minArr2 = arr2[i];
    }
    if (maxArr1 > minArr2)
           return false;
    return true;
}
```

Program 4 The code for the best solution to question 2 part c.

As to part d, we asked the students to write the time and space complexity of their solutions. We found that 91% of the students answered correctly on time complexity, and 72% answered correctly on space complexity. This is a much better result than the results of the previous exams (where only about 28 or 29 percent answered correctly on space complexity). We also found that all of the students who gave the best solution in part c, answered correctly on time complexity, and 85% of them answered correctly on space complexity. Even a significant percentage of the students who gave the less efficient solution, as well as those who gave a wrong solution, knew how to correctly measure the time and space complexities of their solutions - 82% for the time complexity and 57% for the space complexity. We found similar results in the second sitting of the exam.

Pleased this time also with the space complexity results, we decided to proceed with the approach of emphasizing space complexity explicitly, both in face-to-face tutorials and in the exercises.

4. Discussion

Teaching efficiency in CS1 serves as an introduction to a very important topic in computer science. We believe that time should be dedicated not only to analyzing time complexity but also to analyzing space complexity. The students must understand that even though computers are constantly getting smaller, and it seems that the amount of memory needed is no longer an issue, the reasons for learning about space complexity are very similar to those for learning about time complexity, and that both time and space are crucial to almost every use of the computer.

As a result of part one of the study, we changed the way we teach efficiency in our CS1 course. We put more emphasis on space complexity both in face-to-face tutorials and in the exercises. We examined students' perceptions of this concept, and we saw that the change helped them to understand.

One important conclusion we reached was that teachers should not rely on students' ability to make a transfer between two similar concepts by themselves. We have to teach them everything we want them to know.

In future research, we intend to examine students' perception of the concept of space complexity in advanced courses, like data structures and algorithms.

References

1 Du Boulay, B. Some difficulties of learning to program. Journal of Educational Computing Research 2 (1); 1986; 57-73.

2 Gal-Ezer, J., Vilner, T. & Zur, E. Teaching algorithm efficiency at CS1 Level: a different approach. Computer Science Education 14 (3); 2004; 235-248.

3 Gal-Ezer, J. & Zur, E. The concept of 'algorithm efficiency' in the high school CS curriculum. 32nd ASEE/IEEE Frontiers in Education Conference; 2002.

4 Gal-Ezer, J. & Zur, E. The efficiency of algorithms – misconceptions. Computers & Education 42 (3); 2004; 215-226.

5 Ginat, D. Early algorithm efficiency with design patterns. Computer Science Education 11 (2); 2001; 89-109.

6 Ginat, D. Efficiency of algorithms for programming beginners. In Proc of the 27th ACM Computer Science Education Symposium. New York, ACM Press; 1996; 256-260.

7 Harel, D. Algorithmics*: The spirit of computing*, 3rd ed. Reading, MA, Addison Wesley; 2004.

8 IEEE Computer Society/ACM Task Force. Computing curricula *2001* (CC-2001). http://www.computer.org/education/cc200/final .

9 Putman, R. T., Sleeman, D., Baxter, J. A., & Kuspa, L. K. A summary of misconceptions of high school BASIC programmers. Occasional Report No. 10, Stanford and the Schools Project; 1986.

10 Ragonis, N. & Ben-Ari, M. On understanding the static and dynamics of Object-Oriented Programs. In Proc of the 36th ACM Computer Science Education Symposium. ACM Press; 2005; 226-230.

11 Saj-Nicole, A. J., & Soloway, E. But my program runs! Discourse rules for novice programmers. Journal of Educational Computing Research 2 (1); 1986; 95-125.

12 Stemler, L. Effects of instruction on the misconception about programming in BASIC. Journal of Research on Computing in Education; 1989; 26-33.

13 The Open University of Israel website: http://www-e.openu.ac.il/.