Computer Science CSC263H | May 12, 2019
St. George Campus | University of Toronto

Homework Assignment #1

Due: May 26 , 2019, by 11:59pm

- You must submit your assignment as a PDF file of a typed (**not** handwritten) document through the MarkUs system by logging in with your CDF account at:

  `markus.teach.cs.toronto.edu/csc263-2019-05`

  To work with one or two partners, you and your partner(s) must form a group on MarkUs.
- If this assignment is submitted by a group of two or three students, the PDF file that you submit should contain for each assignment question:
  1. The name of the student who *wrote* the solution to this question, and
  2. The name(s) of the student(s) who *read* this solution to verify its clarity and correctness.
- The PDF file that you submit must be clearly legible. To this end, we encourage you to learn and use the LaTeX typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- By virtue of submitting this assignment you (and your partners, if you have any) acknowledge that you are aware of the homework collaboration policy that is stated in the csc263 course web page: `https://github.com/csc263-summer2019/csc263-s19/wiki/Course-Information-Sheet#assignment-collaboration`.
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.

Question 1. (20 marks)

In the following procedure, the input is an array $A[1..n]$ of arbitrary integers, $A.size$ is a variable containing the size n of the array A (assume that A contains at least $n \geq 2$ elements), and "return" means return from the procedure call.

```
void doStuff(int array[], int n){  //n is the size of the array
    array[0]=1;
    int i=1;
    while ( i < n ){
        j=0;
        while(j < i-1){
            array[j]=array[j+1];
            j++;
        }
        if(array[i-1] != array[i]){
            break; //exits the while(i<n) loop
        }
        i++;
    }
}
```

Let $T(n)$ be the worst-case time complexity of executing procedure **doStuff()** on an array A of size $n \geq 2$. Assume that assignments, comparisons and arithmetic operations, like additions, take a constant amount of time each.

a. (5 marks) State whether $T(n)$ is $O(n^2)$ and justify your answer.

b. (15 marks) State whether $T(n)$ is $\Omega(n^2)$ and justify your answer.

Any answer without a *sound and clear* justification will receive no credit.

Question 2. (20 marks)

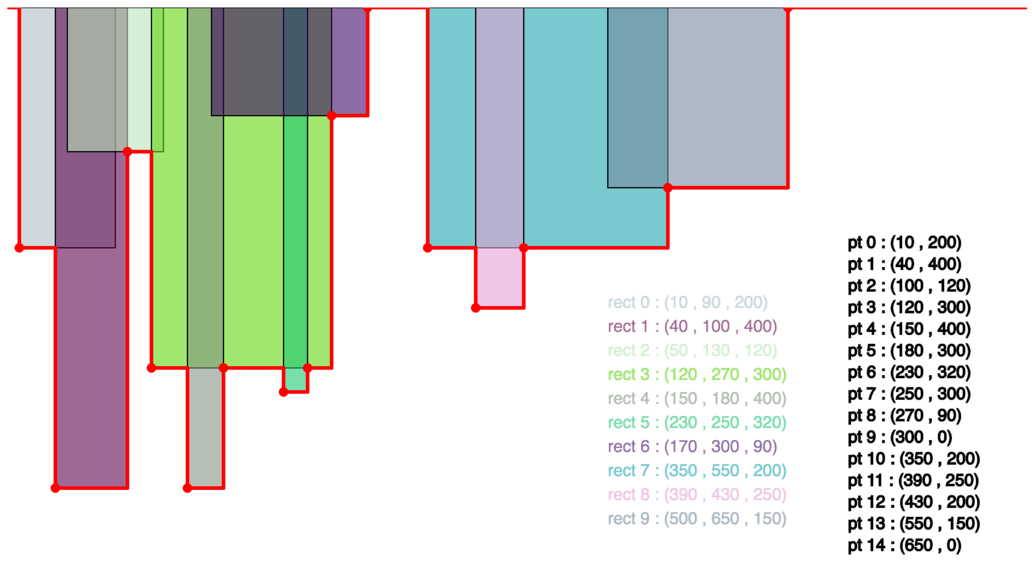
Suppose that we were using a drawing program where the drawing coordinates begin at 0,0 in the top left corner and extends to width-1,height-1 at the bottom right corner. Use this coordinate system as a basis for this question.

We want to find an efficient algorithm for the following problem. The input for the algorithm consists of

- an array of n triples ($x1$, $x2$, h). For simplicity, you may assume that $x1$, $x2$ and h are non-negative integers. You may also assume that the array is sorted in increasing order by the value of $x1$
- Each triple represents a single rectangle that is drawn from the top of the canvas
  - $x1$ is the x position left edge of the rectangle
  - $x2$ is the x position of the right edge of the rectangle
  - h is the height of the rectangle.

The algorithm must output an array of the left end points of the horizontal lines of the shape sorted from left to right. Note that this is all we need to recreate the outline as we can draw starting from 0,0 till we hit the x value of the first end point, draw a vertical line towards it. Once that point is reached we can then continue to the left until we hit the x value of the next horizontal line segment and so on.

For the running example See `https://csc263-summer2019.github.io/csc263-s19/`



Example drawing of rectangles. The dots on the red line and their positions is the output of the function

a. (16 marks) Describe a simple algorithm that solves the above problem. Your algorithm *must* use a data structure that we learned in class, and its worst-case time complexity must be $O(n \log n)$. *Describe the algorithm, and explain why it works, clearly and precisely in plain English.*

b. (4 marks) Explain why your algorithm's worst-case time complexity is $O(n \log n)$.

c. (0 marks) *This section is for absolutely no marks and will not be graded.* It is here only for a sense of self satisfaction. The example graphical web page linked above is created using JavaScript and a drawing library called p5.js. You can find the source code for it inside the docs folder in course repo. The code that does the drawing is in the file called sketch.js. Implement your algorithm by replacing the function setEdges() (starts at line 22) which currently simply hard-codes the solution for the example data.

Question 3. (20 marks) This question is about the cost of successively inserting k elements into a binomial heap of size n.

a. (6 marks) Prove that a binomial heap with n elements has exactly $n - \alpha(n)$ edges, where $\alpha(n)$ is the number of 1's in the binary representation of n.

b. (14 marks) Consider the worst-case total cost of successively inserting k new elements into a binomial heap H of size $|H| = n$. In this question, we measure the worst-case cost of inserting a new element into H as the maximum number of pairwise comparisons between the keys of the binomial heap that is required to do this insertion. It is clear that for $k = 1$ (i.e., inserting one element) the worst-case cost is $O(\log n)$. Show that when $k > \log n$, the *average* cost of an insertion, i.e., the worst-case total cost of the k successive insertions divided by k, is bounded above by constant.

Hint: Note that the cost of each one of the k consecutive insertions varies — some can be expensive, other are cheaper. Relate the cost of each insertion, i.e., the number of key comparisons that it requires, with the number of extra edges that it forms in H. Then use part (a).

[***The questions below will not be corrected/graded. They are given here as interesting problems that use material that you learned in class.***]

Question 4. (0 marks) In class we studied *binary* heaps, i.e., heaps that store the elements in complete *binary* trees. This question is about *ternary* heaps, i.e., heaps that store the elements in complete *ternary* trees (where each node has at most *three* children, every level is full except for the bottom level, and all the nodes at the bottom level are as far to the left as possible). Here we focus on MAX heaps, where the priority of each node in the ternary tree is greater or equal to the priority of its children (if any).

a. Explain how to implement a ternary heap as an array A with an associated $Heapsize$ variable (assume that the first index of the array A is 0). Specifically, explain how to map each element of the tree into the array, and how to go from a node to its parent and to each of its children (if any).

b. Suppose that the heap contains n elements.
(1) What elements of array A represent internal nodes of the tree? Justify your answer.
(2) What is the height of the tree? Justify your answer.

c. Consider the following operations on a ternary heap represented as an array A.

- INSERT(A, *key*): Insert *key* into A.
- EXTRACT_MAX(A): Remove a key with highest priority from A.
- UPDATE(A, i, *key*), where $1 \leq i \leq A.Heapsize$: Change the priority of $A[i]$ to *key* and restore the heap ordering property.
- REMOVE(A, i), where $1 \leq i \leq A.Heapsize$: Delete $A[i]$ from the heap.

For each one of these four operations, describe an efficient algorithm to implement the operation, and give the worst-case time complexity of your algorithm for a heap of size n. Describe your algorithm using high-level pseudo-code similar to that used in your textbook, with clear explanations in English. Express the worst-case time complexity of your algorithm in terms of Θ and justify your answer.

Question 5. (0 marks)

Let I_n be the set of n integers $\{1, 2, \ldots, n\}$ where n is some power of 2.

Note that we can easily use an n-bit vector (i.e., an array of n bits) $B[1..n]$ to maintain a subset S of I_n and perform the following three operations (where j is any integer in I_n) in constant time each:

INSERT(j): insert integer j into S.

DELETE(j): delete integer j from S.

MEMBER(j): return **true** if $j \in S$, otherwise return **false**.

Describe a data structure that supports all the above operations ***and*** also the following operation

MAXIMUM: return the greatest integer in S

such that:

- The worst-case time complexity of operations INSERT(j), DELETE(j), and MAXIMUM is $O(\log n)$ each. The worst-case time complexity of MEMBER(j) is $O(1)$.
- The data structure uses only $O(n)$ bits of storage.

  Note that the binary representation of an integer i where $1 \leq i \leq n$ takes $\Theta(\log n)$ bits. Assume that any pointer also takes $\Theta(\log n)$ bits.

A solution that does not meet ***all*** the above requirements may not get any credit.

HINT: Complete binary trees can be implemented without using pointers.

a. Describe your data structure by drawing it for $n = 8$ and $S = \{1, 2, 6\}$, and by explaining this drawing briefly and clearly. Your drawing must be very clear.

b. Explain how the operations INSERT(j), DELETE(j), and MAXIMUM are executed, and why they take $O(\log n)$ time in the worst-case. Be brief and precise.

c. Explain how the operation MEMBER(j) is executed, and why it takes $O(1)$ time in the worst-case. Be brief and precise.