Please follow the instructions provided below to submit your assignment.

Worth: 10% **Due:** By 11:59 pm on Friday Nov 17

- You must submit your assignment as a single PDF file through the MarkUs system.

  https://markus.teach.cs.toronto.edu/csc263-2017-09/

  The filename must be the assignment's name followed by "sol", e.g. your submission for the assignment "A3" must be "A3sol.pdf". Any group of two students would submit a single solution through Markus. Your PDF file must contain both team member's full names.
- Make sure you read and understand "POLICY REGARDING PLAGIARISM AND ACADEMIC OFFENSE" provided in the course information sheet.
- 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 find a latex template in Piazza under resources. You can use other typesetting systems if you prefer. Handwritten documents are acceptable but not recommended. The submitted documents with low quality that are not clearly legible, will not be marked.
- You may not include extra descriptions in your provided solution. The maximum space limit for each question is 2 pages. (using a reasonable font size and page margin)
- 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.
- For describing algorithms, you may not use a specific programming language. Describe your algorithms clearly and precisely in plain English or write pseudo-code.

1. **Amoritization**

   In this question, we explore a well-known trick that uses two Stacks S_1 and S_2 to simulate the operations of a Queue Q. Consider the following implementations of ENQUEUE and DEQUEUE.

```
ENQUEUE(Q, x):
    if SIZE(S_1) ⩾ 12 and ISEMPTY(S_2):
        while not ISEMPTY(S_1):
            PUSH(S_2, POP(S_1))
    PUSH(S_1, x)

DEQUEUE(Q):
    if ISEMPTY(S_2):
        if ISEMPTY(S_1):
            error "Dequeuing from an empty queue!"
        else:
            while not ISEMPTY(S_1):
                PUSH(S_2, POP(S_1))
    return POP(S_2)
```

   For your analysis, assume that each PUSH operation takes 2 units of time, and each POP operation takes 3 units of time. Each ISEMPTY or SIZE operation takes 0 unit of time. The time taken by any other pseudo-code operation is ignored.

   (a) Consider the sequence of operations that consists of 50 ENQUEUE operations followed by 50 DEQUEUE operations. Use **aggregate analysis** to compute the amortized cost per operation for this sequence.

   (b) Now consider **any** sequence of m ENQUEUE and DEQUEUE operations. Use the **accounting method** to derive an upper-bound on the amortized cost per operation.

2. **Fuzzy sorting of intervals**
   Solve Exercise 7-6 from the textbook (CLRS) page 189.

3. **Randomized Algorithm - Search in an unsorted array**

   The following algorithm uses a randomized strategy to search for a value x in an unsorted array A consisting of n elements.
   The strategy: Choose a random index i into a given array A. If $A[i] = x$, then we return i; otherwise, we continue the search by picking a new random index into A. We continue picking random indices into A until we find an index j such that $A[j] = x$ or until we have checked every element of A. Note that we pick from the whole set of indices each time, so that we may examine a given element more than once.

   (a) Write a pseudocode for a procedure RANDOM-SEARCH to implement the strategy above. Be sure that your algorithm terminates when all indices into A have been picked.

   (b) Suppose that there is exactly one index i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and RANDOM-SEARCH terminates?

(c) Generalizing your solution to part (b), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and RANDOM-SEARCH terminates? Your answer should be a function of n and k.

(d) Suppose that there are no indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we have checked all elements of A and RANDOM-SEARCH terminates?

4. **Randomized Algorithm - Search in an almost sorted array**

An array of n distinct elements with starting index zero is *almost sorted* if the elements are sorted but the smallest index is not necessarily the first index of the array. We are given a reference *head* to the index with smallest element. Clearly, for $head > 0$ the largest element is $A[head - 1]$ and if $head = 0$ the largest element is $A[n-1]$. An example of an almost sorted array would be:

$head = 3$

0	1	2	3	4	5	6	7
33	40	55	5	12	20	25	30

For a given almost sorted array A and its head, consider the following algorithm for performing SEARCH(x):

```
SEARCH(A, head, x)
    Uniformly choose i ∈ {0,...,n−1}.
    Compare A[i] and x:            (The number of comparisons in each step is one, not three)
        If A[i] = x      return i
        If A[i] > x      i ← head
        If A[i] < x      i ← ((i + 1) mod n)
    While True do
        Compare A[i] and x
            If A[i] = x      return i
            If A[i] > x      return -1
            If A[i] < x      i ← ((i + 1) mod n)
            If i = head      return -1
```

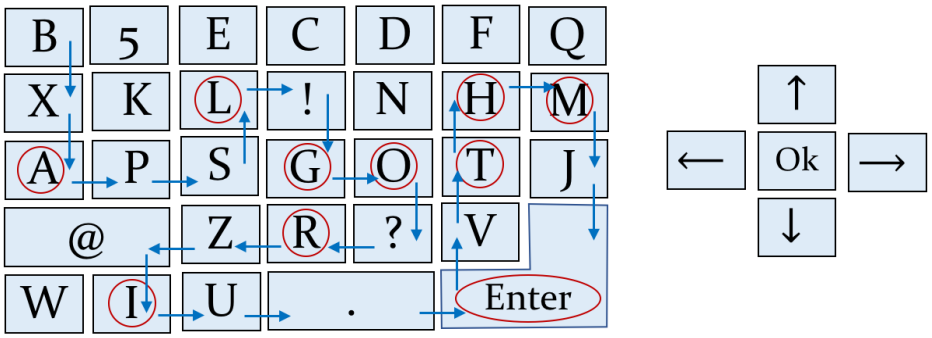
(a) What is the worst case number of comparisons with x performed by SEARCH(x)? Briefly justify your answer.

(b) Determine the expected worst case number of comparisons with x performed by SEARCH(x):

- Define the sample space.
- Define the random variables for your analysis.
- Compute the expected worst case number of comparisons with x performed by SEARCH(x) and then, write your solution in terms of big-O and Ω.

5. In this question, we want to see how many keystrokes are required to type a text message? You may think that it is equal to the number of characters in the text, but this is correct only if each keystroke generates only one character. With pocketsize devices, the possibilities for typing text are often

Figure 1: Sample Input. An example for virtual keyboard and hardware buttons. It shows a possible way to type ALGORITHM using 32 strokes on an example virtual keyboard.



limited. Some devices provide only a few buttons, significantly fewer than the number of letters in the alphabet. For such devices, several strokes may be needed to type a single character.

One mechanism to deal with these limitations is a virtual keyboard displayed on a screen, with a cursor that can be moved from key to key to select characters. Four arrow buttons control the movement of the cursor, and when the cursor is positioned over an appropriate key, pressing the fifth button selects the corresponding character and appends it to the end of the text. To terminate the text, the user must navigate to and select the Enter key. This provides users with an arbitrary set of characters and enables them to type text of any length with only five hardware buttons.

In this problem, you are given a virtual keyboard layout in a two dimensional array A with r rows and c columns. Your task is to determine the minimal number of strokes needed to type a given text, where pressing any of the five hardware buttons constitutes a stroke. Note that you stroke OK key in the middle to type the letter which **should be counted**. The keys are arranged in a rectangular grid, such that each virtual key occupies one or more connected unit squares of the grid. The cursor starts in the **upper left corner of the keyboard** and moves in the four cardinal directions, in such a way that it always skips to the next unit square in that direction that belongs to a different key. If there is no such unit square, the cursor does not move.

Figure 1, shows an example for a virtual keyboard. For such an input, we are given an array of size 7×5. It shows a possible way to type ALGORITHM using 32 strokes. Note that there is only one key corresponding to any given character. Each key is made up of one or more grid squares, which will always form a connected region. The character key '$\backslash n$' represents Enter in the virtual keyboard. The input is A and s where A is an array of size $r \times c$ as a virtual keyboard and s is a string of size n as an input text to be typed.

$$A = \begin{bmatrix} B & 5 & E & C & D & F & Q \\ X & K & L & ! & N & H & M \\ A & P & S & G & O & T & J \\ @ & @ & Z & R & ? & V & \backslash n \\ W & I & U & . & . & \backslash n & \backslash n \end{bmatrix}$$

(a) Write a pseudocode for an algorithm that finds the minimal number of strokes necessary to type the whole text, including the Enter key at the end of typing. It is guaranteed that the text can be typed.

(b) Analyze the running time of your algorithm.