COMP 3711 – Spring 2019
Tutorial 2

1. Derive asymptotic upper bounds for $T(n)$. Make your bounds as tight as possible.

   (a) You may assume n is a power of 2.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + n \qquad \text{if } n > 1 \end{aligned}$$

   (b)

   $$\begin{aligned} T(1) &= T(2) = 1 \\ T(n) &= T(n-2) + 1 \qquad \text{if } n > 2 \end{aligned}$$

   (c) You may assume n is a power of 3.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/3) + n \qquad \text{if } n > 1 \end{aligned}$$

   (d) You may assume n is a power of 2.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(n/2) + n \qquad \text{if } n > 1 \end{aligned}$$

   (e) You may assume n is a power of 2.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(n/2) + n^2 \qquad \text{if } n > 2 \end{aligned}$$

   (f) You may assume n is a power of 2.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= T(n/2) + \log_2 n \qquad \text{if } n > 1 \end{aligned}$$

   (g) You may assume n is a power of 2.

   $$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(n/2) + n \log_2 n \qquad \text{if } n > 1 \end{aligned}$$

2. Input: a sorted array $A[1..n]$ of n **distinct** integers
(Distinct means that there are no repetitions among the integers. The integers can be positive, negative or both).

Design an $O(\log n)$ algorithm to return an **index i such that $A[i] = i$,** if such an i exists. Otherwise, report that no such index exists.

As an example, in the array below, the algorithm would return 4.

i	1	2	3	4	5	6	7	8
A[i]	-3	0	1	4	12	17	20	22

while in the array below this line, the algorithm would return that no such index exists.

i	1	2	3	4	5	6	7	8
A[i]	-3	0	1	7	12	17	20	22

Hint: $O(\log n)$ often denotes some type of binary search. Can you think of any question you might ask that permits you to throw away some constant fraction of the points

3. Let $A[1..n]$ be an array of n elements. A *majority element* of A is any element occuring more than $n/2$ times (e.g., if $n = 8$, then a majority element should occur at least 5 times). Your task is to design an algorithm that finds a majority element, or reports that no such element exists.

   (a) Suppose that you are not allowed to order the elements; the only way you can access the elements is to check whether two elements are equal or not. Design an $O(n \log n)$-time algorithm for this problem.

   (b) A more advanced question is to design an $O(n)$ algorithm for this problem. Similar to (a), you are still only allowed to use equality tests on the elements.

   Note: Obviously a solution to (b) is also a solution to (a). The intent here is to use standard divide-and-conquer techniques to solve (a). (b) is extra and permits getting a faster solution in (b) by using more problem specific properties. At this point, you are not expected to be able to solve (b).

4. **Divide and Conquer**

   You have found a newspaper from the future that tells you the price of a stock over a period of n days next year. This is presented to you as an array $p[1 \ldots n]$ where $p[i]$ is the price of the stock on day i.

   Design an $O(n \log n)$-time divide-and-conquer algorithm that finds a strategy to make as much money as possible, i.e., it finds a pair i, j with $1 \leq i < j \leq n$, such that $p[j] - p[i]$ is maximized over all possible such pairs.

   If there is no way to make money, i.e., $p[j] - p[i] \leq 0$ for all pairs i, j with $1 \leq i < j \leq n$, your algorithm should return "no way".

   (a) Describe your algorithm and explain why it gives the correct answer

   (b) Analyze your algorithm to show that it runs in $O(n \log n)$ time.

5. The following is the pseudo-code for a procedure known as *BubbleSort* for sorting an array of n integers.

```
repeat    % start new pass
    swapped := false;
          for i = 1 to n-1
               if A[i] > A[i+1]  then
                  swap A[i] and A[i+1]
                  swapped := true
until  not(swapped)
```

   A *pass* in the algorithm is a walk from left to right in the array, comparing each item to its successor and, if they are in the wrong order, swapping them. The algorithm repeats this process until it encounters a pass in which no items are swapped.

   (a) Prove that Bubble sort terminates and, at termination has correctly sorted its input.

   (b) What is the worst-case input for bubble sort? Use it to derive a lower bound on the time complexity of Bubblesort in the worst case.