ADS
Homework 4

Samundra karki

March 12, 2019

Problem 4.1

(a)

The pseudo code for the bubble-sort algorithm is given below:

PSEUDO-CODE FOR BUBBLE-SORT ALGORITHM

```
    """this code will repeat until swapped is happenening in the loop.
    If swap function is not called that means A[i-1] is not > A[i],
    that means array is already soreted. We dont need to loop further.
    We are taking index starting position 0"""

Bubble-sort(A, n): #Buble sort function takes the array and
                     size as its parameter
    repeat
        swapped = false
        for i = 1 to length(A)-1
            if A[i -1] > A[i] then
                #if element having A[0] > A[1], then swap function is called
                #leading swapped keyword to be true
                swap(A[i-1], A[i])
                swapped = true
            endif
        endfor
    until not swapped

    """unitl swapped is not happening in the list this
    pesudo-code will loop in order to sort the array"""
```

(b)

Let's see the time complexity of Bubble-sort:

$$T(n) = c_1 n + c_2(n-1) + c_3\sum_{j=1}^{n} t_j + c_4\sum_{j=1}^{n}(t_j - 1) + c_5\sum_{j=1}^{n}(t_j - 1)$$

Worst case:

Here, in worst-case, the array is in reverse order and outer loop iterates from 1 to n and inner loop iterates 2 to n. However, the looping statement will iterate 2 to n+1 to fail the loop. So, $t_j = n$ can be written in this condition.

At each step in the inner loop in the iterating condition. the loop iterates 2 to n+1 times i.e. n times. so it can be written as $n + n + n + ... + n$

$$\sum_{j=1}^{n} j = \sum_{j=1}^{n} n = n^2$$

We can see that $c_3\sum_{j=1}^{n} t_j$ is the term with the highest power as terms right of it is iterating one times less and left terms cannot lead to n^2. So, this is a quadritic function.

So the worst case is O(n^2) as derived from mathematical proof which can be easily figured out by the outer and outer loop.

Best case:

Here, in the case of best case, array is already sorted. That means, though the second loop iterates n+1 times but the swapped is not done. And the repeat is also not carried out as at first iteration swapped is not done which means swapped is false.

Mathematically: $t_j = 1$

$$\sum_{j=1}^{n} 1 = n$$

Every term will be constant time n, which will lead to a linear function. Thus time complexity is O(n).

Average case:

Let's say that the at average case loop iterates till $\frac{n}{2}$. However, the inner loop iteration condition statement is iterating n times. So, we can say that:

$$\sum_{j=1}^{\frac{n}{2}} j = \sum_{j=1}^{\frac{n}{2}} n = \frac{n^2}{2}$$

This still is a quadratic equation, so this as it is the highest order term in this function. So, the average run time is O(n^2).

(c)

To make this problem simpler let's see a array with following sequence:
16 14 10(a) 2 3 5 10(b) 4
10(a) and 10(b) represents which 10 is indexed first in the array.

Insertion Sort:

This sorting algorithm first takes second emelent of array as key. Then it compares it with the elements prior to it. If it finds that the array indexed at position less than key is greter than key then it interchanges the value of the key. So first 14 is kept at index 1. Then 10(a) 14 16 is placed in sorted array. Then according to the following sequence the process is carried out:

- 2 10(a) 14 16
- 2 3 10(a) 14 16
- 2 3 5 10(a) 14 16
  Now according the alogrithm, only if A[i] > key, then swapping is done.
  So, in this case as 10(b) reaches to the position to check if A[i] > key. It fails. So:
- 2 3 5 10(a) 10(b) 14 16
- 2 3 4 5 10(a) 10(b) 14 16

As, the 10(a) appearing before then 10(b) list also appear in same order after the sorting is done. So, the this alogorithm is stable.

Merge Sort:

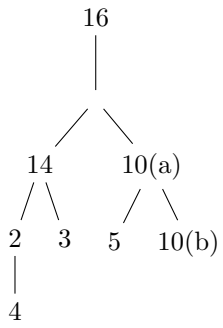
This algorithm is stable. As, merge sort first divide the problem into individual sub-problem by recurrsion and then merge it together. If we see in the given example, then each element of the array will treat itself as a array contaning single element which is always sorted. Then while merging this condition should be met:

```
if L[i] <= R[i] then
    A[k] = L[i]
```

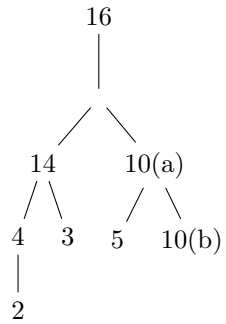
If 10(a) and 10(b) meets the condition then in the array 10(a) is placed at first. So, this alogrithm is stable.

Heap sort:

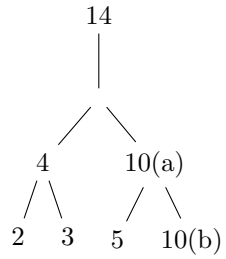
Heap sort will make a nearly complete binary tree. We can visulize it as:



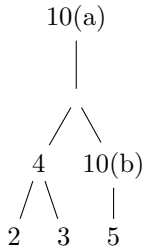
So in first case, to satisfy max-heap property, following binary tree is acheived.



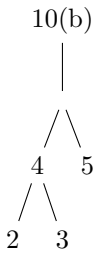
After max-heap is created, we are going to take the max element which is at A[1].Then take it to last element of the heap then deattach it from the tree making it the last elemet of the sorted array. After that:



After 14 is removed, 10(a) gets into the top of the tree as it will be the greatest element.Then gets pushed in sorted array as 10(a) 14 16.



After that, 10(b) gets pushed in the array so, 10(b) appears first in the order. This voilates the stability rule. So heap sort is not stable.



Following the steps, the sorting algorithm would take us to:
2 3 4 5 10(b) 10(a) 14 16

Bubble-Sort:

As, algorithm is already explained. We can clearly see why it is stable. Only when the A[i-1] > A[i] then swapping occurs. However, we are not bothered with when they are equal so, we dont change(swap) them when they are equal. So, bubble sort algorithm is stable.

Following the steps, the sorting algorithm would take us to:
2 3 4 5 10(a) 10(b) 14 16

(d)

As, I have explained hoe all the algorithm works in the previous question, so finding whether a algorithm is adaptive or not is rather easy.

Insertion sort

This algorithm takes the advantage of the pre-sorted input and performes faster when the input is sorted i.e. O(n). As, we know that it would insert and swap the element in its ordering position, so, if the input is pre-sorted then,inner loop is never enterred and swap() fucntion would not be called, and the time taken would be less.

Merge sort

This algorithm is failing to take advantage of the pre-sorted input and takes same time as any unsorted array i.e O($nlogn$). This is because first it takes the input and divides it until single element are passed into merge function. So, the recursive calls did not look for if the input is already sorted or not, it just divided the input into sub-problems. So, we can state the merge-sort is not adptive.

Heap sort

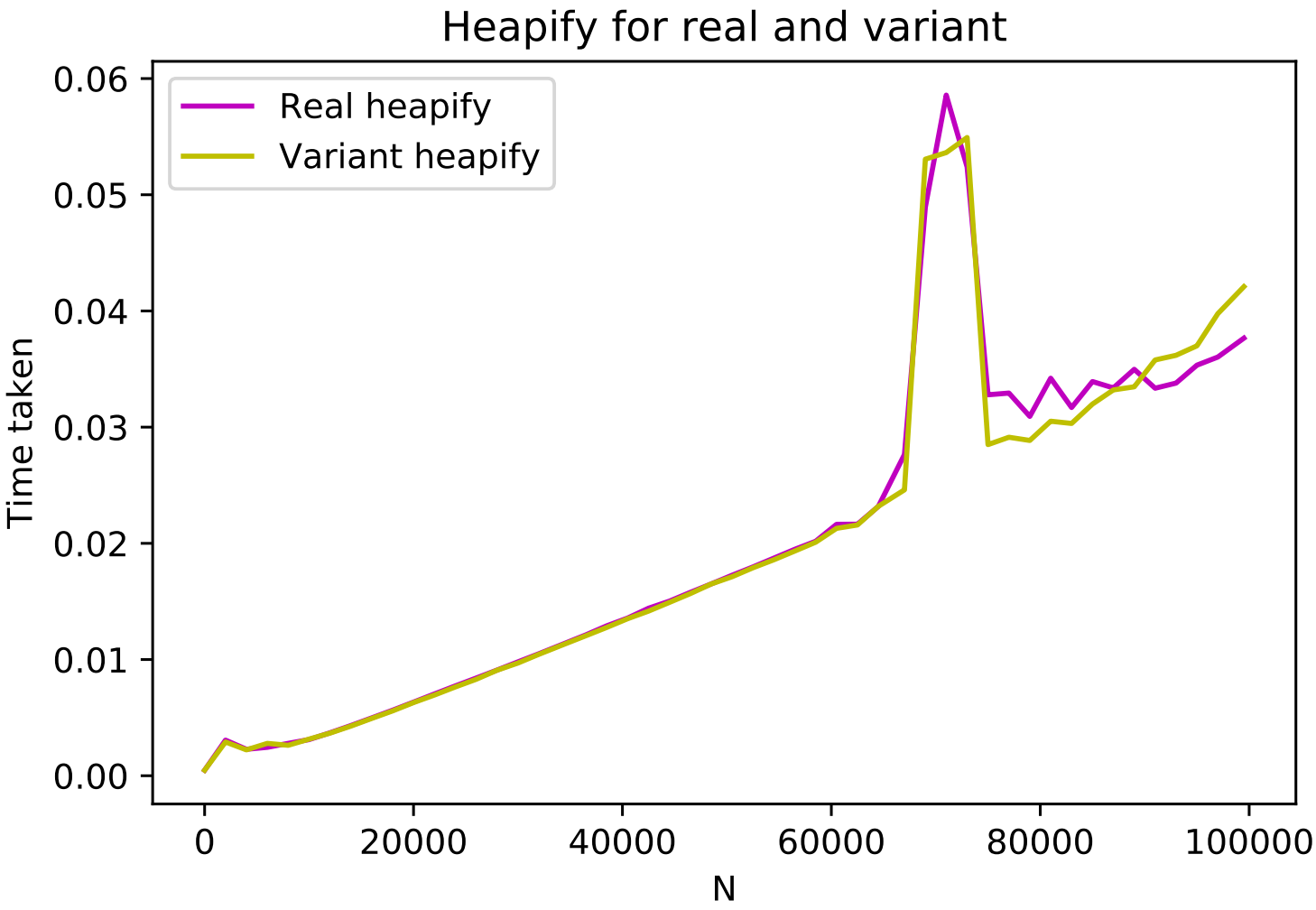
I have explained how the heap-sort works in **problem 4.1 c**. This algorithm takes O($nlogn$) for best case and for the worst case. It doesnot take advantage of the pre0-sorted input and takes same time as unsorted array. It can be seen because it first converts the input into max-heap, if it is already max-heap then it swaps max element with the last indexed element and breaks the link. So, this will carry on again as the last element would not be essentially be second max-element if the array is pre-sorted too. So this algorithm is not adpative.

Bubble-sort

This algorithm is adpative. As, we have programmed the algorithm to run only if swapped is true, that means, if the input is pre-sorted than the algorithm then it would falsify the swapped as the swapped will not be enetered, so the loop ends, making the time complexity O(n).

Problem 4.2

(c)


Heapify for real and variant
Real heapify
Variant heapify
Time taken
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
20000
40000
60000
80000
100000
N

Here, this graph shows that time taken by heap variant and heap sort is not different by so much. However, if the node that was pushed down to the leaves was actually greater than its parent then it should be pushed up again, so heap variant should have taken more time giving the time complexity more than the heap sort.
The values taken in the program were random and and time complexitity was not different due to the fact that the numbers genereted by the random number generator supported the fact the numbers that was taken to leaves were actually smaller values so the time complexity of both programs were not significantly different.