ECE 9343 - Homework Unit-4

Data Structure and Algorithm (New York University)

[redacted]

DSA HW-4

1. Initial Array, (First Iteration)

$A = \langle 14, 12, 14, 19, 5, 3, 4, 14, 7, 22, 16 \rangle$

The Hoare's Partition algorithm involves selecting a pivot element and rearranging the elements such that all elements less than the pivot are on the left and all elements greater than the pivot are on the ~~left~~ right of the pivot element.

Let pivot, $x \leftarrow A[0] \Rightarrow x = 14$

	p										r	
	14	12	14	19	5	3	4	14	7	22	16	
↑i												↑j

After following instructions at line 5,

14	12	14	19	5	3	4	14	7	22	16
↑i								↑j		

After following instructions at line 7,

14	12	14	19	5	3	4	14	7	22	16
			↑i					↑j		

Since, $i < j$, exchanging 19 & 7,

14	12	14	7	5	3	4	14	19	22	16
			↑i					↑j		

Reiterating through the loop, (line 5 ~~ite~~ instruction)

14	12	14	7	5	3	4	14	19	22	16
			↑i			↑j				

Now, following instructions at line 7,

14	12	14	7	5	3	4	14	19	22	16
						↑j		↑i		

Since $i > j$ we will return j that is the index value of array element 4 which is 6.

$\boxed{j=6}$ → Returned as the partion index.

2(a) Original array : A=< 3, 9, 5, 8, 15, 7, 4, 10, 6, 12, 16 >

(a) To build a max heap using the BUILD-MAX-HEAP algorithm. The idea is to start with the last non-leaf node and perform MAX-HEAPIFY on each node in reverse order until we reach the root of the heap.

Identify the last non-leaf node using floor(n/2) where n = size of the array.

Starting from A[5] and moving in reverse order to index 0, apply MAX-HEAPIFY on each node.

Start with index 5. Compare with children (A[11] and A[12]). Swap A[5] with the larger child which is 16. ~~No swapping is required~~ The array will be transformed into,

A= < 3, 9, 5, 8, 15, 16, 4, 10, 6, 12, 7 >

Moving to index 4 and repeating the intsuctions, No swap is required. There is no change in the array.

Moving to A[3]. Compare with children and swapping with larger child.

→

The array will now look like this,

$$A = \langle 3, 9, 5, 10, 15, 16, 4, 8, 6, 12, 7 \rangle$$

Continuing the process by applying MAX-HEAPIFY on all nodes we get the following array,

$$A = \langle 16, 15, 5, 10, 12, 7, 4, 8, 6, 3, 9 \rangle$$

2(b) Removing the largest item from the max heap using HEAP-EXTRACT-MAX.

~~After~~ Remove the largest element (root element) and replace it with the last element in the heap (9). This will temporarily break the max heap property

Initial array :- A = <16, 15, 5, 10, 12, 7, 4, 8, 6, 3, 9>

After removing the largest element, (after removal and replacing with the last element)

A = <9, 15, 5, 10, 12, 7, 4, 8, 6, 3>

Now applying MAX-HEAPIFY on root element 9 we get,

A = <15, 9, 5, 10, 12, 7, 4, 8, 6, 3>

This maintains the max heap property.

2.(c) Using the MAX-HEAP-INSERT algorithm, insert 11 into the heap from step 2(b).

We first add 11 to the array and then apply MAX-HEAPIFY to restore the max heap property. Here's how the array looks after inserting 11,

Array A = < 15, 12, 7, 10, 9, 5, 4, 8, 6, 3, 11 >

Now applying MAX-HEAPIFY,

MAX-HEAP A = < 15, 11, 5, 10, 12, 9, 4, 8, 6, 3, 7 >

3. To find the median of an disordered array with n elements by using 2 heaps. Max heap to store smaller half of the elements and min heap to store the larger half.

Algorithm:-

1) Initialize an empty max heap (left heap) to store the smaller half of the elements and an empty min heap (right heap) to store the larger half.

2) Traverse the array one element at a time. As you receive each element, compare with the current median (root). If element is less than or equal to current median add it to the max heap otherwise add it to the min heap.

3) After inserting the element, ensure that the sizes of the left and right heaps are balanced. The max difference can be 1. If the heap is not balanced then perform heap balancing operation. i.e. if the left heap has more elements, move the root of the left heap to the right heap and vice versa.

4) The median of the array can be determined based on the roots of the left and right heaps:-

a) If the sizes of the heaps are equal, the median is the average of the roots of the left and right heaps.

b) If the left heap has more elements, the median is the root of the left heap.

c) If the right heap has more elements, the median is the root of the right heap.

5) Continue this process for all elements in the array.

At the end of the traversal, you will have maintained two heaps that store the smaller and larger halves of the array, allowing you to efficiently find the median.

The time complexity of the algorithm is $O(n*\log(n))$ due to the heap operations, and it required only single traversal of the array.

4.(a) When using HOARE-PARTITION as the PARTITION function in the QUICKSELECT algorithm, the worst case running time occurs when each partition is extremely unbalanced. This can happen when the pivot chosen in each step consistently leads to partition of size (n-1) and 0. In this worst-case scenario, the algorithm performs poorly.

To analyze the worst case running time, we can model it using the recurrence relation:

$$T(n) = T(n-1) + O(n)$$

In each step, the algorithm reduces the problem size by 1, and the partition step takes $O(n)$ time. In the worst case, the algorithm may perform n partitions, and the partitioning becomes very unbalanced.

Solving the recurrence relation:

$$T(n) = T(n-1) + O(n)$$
$$= T(n-2) + O(n-1) + O(n)$$

Continuing this pattern, we get:

$$T(n) = T(n-k) + O(n-(k-1)) + O(n-(k-2)) + \dots + O(n)$$

Until we reach $T(1)$, where the problem size becomes 1, and we can assume that it takes constant time $O(1)$ to solve.

$$T(1) = O(1)$$

Now, let k be be the number of recursive calls until we reach $T(1)$:

$$n = 1 + 2 + 3 + \dots + k$$

Using the arithmatic series sum formula:

$$n = \frac{k(k+1)}{2}$$

Solving for k:

$$k^2 - 2n + k = 0$$

Using the quadratic formula:

$$\boxed{k = \frac{-1 + \sqrt{1+8n}}{2}}$$

In the worst case, k grows as square root of n.
Therefore, the worst case running time of QUICKSELECT with HOARE-PARTITION is $O(n^{3/2})$

4.(b) When utilizing BFPRT (median-of-medians) algorithm as partition in QUICKSELECT, the worst case running time is improved compared to HOARE-PARTITION. BFPRT ensures that the pivot chosen is relatively close to the true median, which helps balance the partitions.

The worst case running time of QUICK-SELECT with BFPRT as PARTITION is O(n), because BFPRT guarantees that the pivot is within constant factor of the true median, leading to balanced partitions in each step. The recursive depth is logarithmic, and each leading to balanced partitions in each step. The recursive depth is logarithmic and each level takes linear time, resulting in an overall worst-case time complexity of O(n).