Student: Tran Ngoc Son (陳玉山)
ID: 0710185

Homework 3 report

Contents

Student: Tran Ngoc Son (陳玉山)
ID: 0710185

I. Introduction

- In this homework, we are going to implement a heapsort algorithm by using Max-heap. First, we have to code the function called “Build max-heap”, which can turn an array becomes Max-heap array structure. Inside Build max-heap function, there is also a function called “Max-heapify”, this function is to make sure a parent and child are meet the “Max-heap” property. Lastly, we make use of the above function to code a function called Heap-sort, this function will help to sort the array. All of the aforementioned functions will be discussed in the methodology section.

II. Methodology

- By the definition, a heap is an array object that can be viewed as a nearly complete binary tree. Max-heap must include root, parents, and child, each parent will have two children, and the value of them is larger than their child. Below are some of the properties of Max-heap (we assume the array index starts at 1 in this report).
    - Root: A[1]
    - Parent of A[i] = A[round-down(i/2)]
    - Left child of A[i] = A[2i]
    - Right child of A[i] = A[2i+1]
    - Height of heap: log(n)
- There are two types of heap, max-heap, and min-heap. Max-heap is the type that parents must have a value greater or equal to their child. In contrast, Min-heap is the one that has parents smaller than their children. Based on their property, we may apply it in different circumstances. However, we will use Max-heap in this report in order to have a deeper understanding of this max heap-sort.

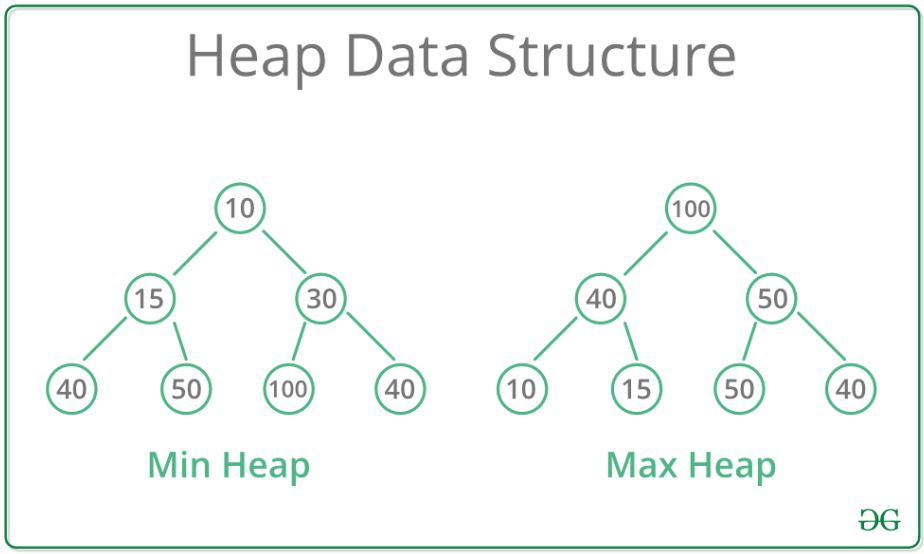

Figure 1: Min Heap and Max Heap example (source: geeksforgeeks.org)

Student: Tran Ngoc Son (陳玉山)
ID: 0710185

- In Max-heap algorithm, the most essential function is Max-heapify(A,i). This function will take two parameters, A is an array, i is the index of the parent. When Max-heapify is called, it will help the passed in parent and child to meet the max-heap property that the parent's value is always larger or equal to the child's value. Below is the pseudo-code of Max-heapify.

Max-heapify(A,i)

```
l = left (i)  // l is index
r = right (i) // r is index
largest = i      ↗ This is to stop compare if l or r is go out of range, otherwise, the
                   algorithm won't work properly
if   l < arr. size  and  arr[l] > arr [i]
     largest = l

if  r < arr . size  and  arr [r] > arr [largest]
     largest = r

if  largest ≠ i  → note this condition
     exchange ( arr [ largest ] , arr [i] )
Max - heapify (A, largest)
```

Figure 2: Pseudo codes of Max-heap

- Basically, this function is going to compare the value of A[i] (parent) and A[2i], A[2i+1] (child) to find out the largest value. After comparing, it will execute the exchange process to let the largest one be the parent.
- After coding the above function, we can implement the "Build-max heap" function. "Build max-heap" will take advantage of the "Max-heapify" function to build the max-heap array. We start to use the function Max-heapify from A.length/2, which means that all parents can be covered since the parent of A[A.length] is A[A.length/2].

Build max-heap (A)

```
A . heap - size  =  A . length
   for   i = ⌊ A . length /2 ⌋ down to 1
        Max - heapify ( A , i )
```

Figure 3: Build max-heap function

- Finally, we can have a sorted array by calling “heap-sort” function. When the Heapsort is called, it will first build the heap-max array by calling “build-max heap”. We may also give the max-heap the size of an array after exchanging, so that the function does not need to deal with the maximum value that has just been exchanged to the last.

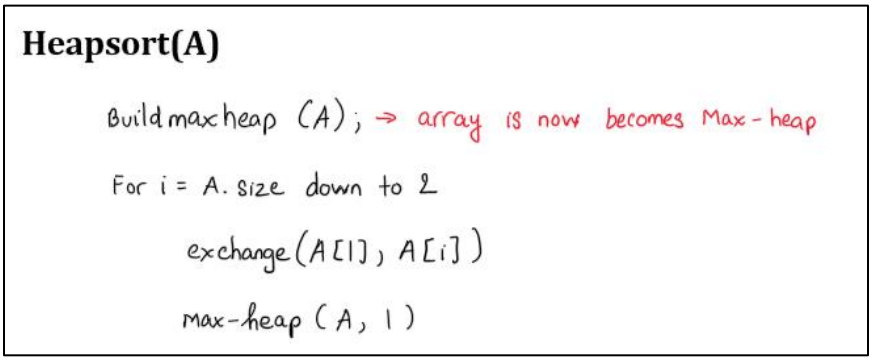

Figure 4: Heapsort pseudo code

III. Results

1. Sort results

- Below is the output of unsorted array and sorted array.

```
Unsort array = 2 0 3 4 8 4 9 3 5 2

Sorted array = 0 2 2 3 3 4 4 5 8 9
```

Figure 5: Unsort and Sorted result

2. Big O discussion

- The running time of Max-heapify is O(log(n))
- The running time of Build-maxheap is O(n)
- Therefore, the total running time of Heapsort should be O(nlog(n))

3. Running time comparison

- From figure 6, we can see the comparison of the running time between the three sorting methods, Heapsort, Mergesort, and Insertion sort. The results show that Heapsort sends the fastest running time most of the time except for the input size of 500. Since the time complexity of Heapsort and Mergesort is O(nlogn), there are some cases the results are quite similar between these two methods, n = 100, 3000, 5000, and 10000. Insertion sort is the one that has the slowest performance in most cases, especially when the input size becomes significantly large. When the input sizes are small, less than 100, the three methods give out the running time quite impressively fast and almost identical results. In this experiment, with the input size of 100 000, heapsort has the best performance and the worst goes for insertion sort.

Student: Tran Ngoc Son (陳玉山)
ID: 0710185

Input size	Heapsort (microseconds)	Mergesort (microseconds)	Insertionsort (microseconds)
100	0	0	0
500	1035	0	1502
700	0	999	3033
1000	524	997	3018
2000	0	2036	21913
3000	1995	2312	50257
5000	4042	4029	156370
7000	2992	5024	249324
10000	16410	16392	1006541
100000	41170	63976	45758476

IV. Discussion

- From this homework and the comparison between the three methods, I realize that even though the algorithm has the same bigO notation, O(nlog(n)) for example, it's the running time would be different for different cases and input sizes.