Algorithms and Data Structures

Arnav Singh

March 2020

6 Sheet 6

6.1 Bubble Sort Stable and Adaptive Sorting

(a) Bubble Sort

Algorithm 1 Bubble Sort

```
1:  function BubbleSort(A, n )        ▷ Where A - array, n - size of array
2:      for i = 1 to n − 1 do
3:          Swapflag==false
4:          for j = 1 to n − i − 1 do
5:              if A[j] < A[j + 1] then
6:                  Swapflag==true
7:                  Swap(A[j], A[j+1]) ▷ Exchanging (j+1)th element with jth
8:                  if Swapflag==false then break
9:
10:
```

▷ The first for loop runs till the number of pass which is given by n-1
▷ The second for loop is for looping through every pass and checking if the next element is greater than the previous if it is then it swaps else it continues.Note that in every iteration the loop runs for one less element because the last element is the greatest element and doesn't need to be checked again.

(b) Time Complexity

Worst Case
The worst case for the Bubblesort would be if the given array is in reverse order.As the outer loop increases the inner loop will perform -1 comparisons after every iteration. So for the first iteration, the number of comparisons and the number of swaps will be n-1, since every comparison leads to a swap. As the outer loop increases the compairosns and swaps will decrease by 1. Hence we have,

$$T(n) = O[(n-1) + (n-2) + (n-3) + ... + 1]$$

This is an arithmetic series and the sum is by Gauss Formula

$$= O(\tfrac{n(n-1)}{2})$$

$$= O(n^2)$$

Average Case
Now for the average case lets assume half of the elements are swapped, therfore the number of comparisons would be

$$= O(\tfrac{n(n-1)}{2*2})$$

$$= O(n^2)$$

Best Case
The best case would be when the array is sorted.When the array is sorted the algorithm performs just (n-1) comparisons and runs for only one iteration since the boolean no. of Swaps becomes false and breaks out of the loop.

$$= O(n)$$

(c) Stability of various Algorithms

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Insertion Sort Insertion sort is a stable algorithm, Initially consider the case we have an unsorted array

0	1	2	3	4	5
10	4_a	4_b	5	2	7

After applying insertion sort we get

0	1	2	3	4	5
2	4_a	4_b	5	7	10

Notice here the positions of the same key value pair i.e 4 is still the same i.e 1 and 2.Thus the algorithm is said to be stable.

Merge Sort Merge sort is a stable algorithm, Initially consider the case we have an unsorted array

0	1	2	3	4	5
10	4_a	4_b	5	2	7

After applying Merge sort we get

0	1	2	3	4	5
2	4_a	4_b	5	7	10

Notice here the positions of the same key value pair i.e 4 is still the same i.e 1 and 2.Thus the algorithm is said to be stable.

Heap Sort

Heap sort is a stable algorithm, Initially consider the case we have an unsorted array

0	1	2	3	4	5
10	4_a	4_b	5	2	7

After applying Heap sort we get

0	1	2	3	4	5
2	4_b	4_a	5	7	10

However, notice here the positions of the same key value pair i.e 4 is not the same they change ,i.e 4b is now at position 1 and 4a is now at position 2. Thus the algorithm is not stable.

Bubble Sort

Bubble sort is a stable algorithm, Initially consider the case we have an unsorted array

0	1	2	3	4	5
10	4_a	4_b	5	2	7

After applying bubble sort we get

0	1	2	3	4	5
2	4_a	4_b	5	7	10

Notice here the positions of the same key value pair i.e 4 is still the same i.e 1 and 2.Thus the algorithm is said to be stable.

(d) Adaptiveness

If order of the elements to be sorted of an input array matters (or) affects the time complexity of a sorting algorithm, then that algorithm is called an *"Adaptive"* sorting algorithm. Hence we take a look at the time complexity of the algorithms.

Insertion Sort

Best Case: $O(n)$

Worst Case:$O(n^2)$

There is a change in the time complexity of the algorithm which means the algorithm takes advantage of the input given , hence the algorithm is **adaptive**.

Merge Sort

Best Case: $O(nlogn)$
Worst Case:$O(nlogn)$
There is no change in the time complexity of the algorithm which means the algorithm takes same time no matter the given input , hence the algorithm is **not adaptive**

Heap Sort

Best Case: $O(nlogn)$
Worst Case:$O(nlogn)$
There is no change in the time complexity of the algorithm which means the algorithm takes same time no matter the given input , hence the algorithm is **not adaptive**

Bubble Sort

Best Case: $O(n)$
Worst Case:$O(n^2)$
There is a change in the time complexity of the algorithm which means the algorithm takes advantage of the input given , hence the algorithm is **adaptive**

6.2 Heap Sort

(a) Implemented check heapsort.cpp in the zip file

(b) Implemented check variant.cpp in zip file

(c) Bonus

The graph shows that as the input size of the array increases the variant takes a decreasing amount of time than the heapsort. This implies for larger size of n the variant will take less time. This is primarily because the variant does only one comparison per level, as the node that was exchanged sits at the top of the heap, it does one comparison per level and finds the path of the largest child to the leaf levels.From there on it checks upwards if what it swapped was right,if not the node is swapped with the parent. This way saves time as in the original heap-sort the max-heapify function requires two comparisons, to find the minimum of the two children and the node.