Analysis of Algorithm I

REPORT

Project 3

MEHMET BARIŞ YAMAN – 150130136

27.11.2016

PART A

QUESTION 1

The running time of Radix Sort with respect to different input numbers are given in the table below.

Algorithm / Number of Inputs	1	100	10000	1000000	100000000
Radix Sort	8.2e - 05	0.005265	1.7e - 05	0.474374	138.37

TABLE 1 — Running times for Radix Sort Algorithm

In general, the running time of Radix Sort is O(n*k) and k is the input size. By inspecting the results, we can see that running times depend on n, since the input size, that is the digits that each number in the text file is including, is not changed. So, the increment in running time likes Insertion Sort algorithm as expected.

QUESTION 2

In general, the algorithm which is stable can be used as an intermediate in Radix Sort Algorithm. Therefore, we must find out which is stable algorithm.

Stability in algorithms can be measured by looking at if the inputs in the same key are ordered as same in output or not.

Firstly, we need to check whether Freezing Sort algorithm is stable or not by giving example inputs 1, 2a and 2b. So, the array includes 3 elements including 1, 2a and 2b in order. Therefore, n = 3 in the freeze sort algorithm.

```
for (int freeze_count = 0; freeze_count < n; freeze_count++){

        for (int i = freeze_count; i < (n + freeze_count) / 2 +1; i++) {

                if (i != n - i + freeze_count && arr[i] < arr[n - i + freeze_count]) {

                        int temp = arr[n - i + freeze_count];
                        arr[n - i + freeze_count] = arr[i];

                        arr[i] = temp;

                 }

                //In the first loop, it changes the places of 2b and 1
                //In the second loop it does not change any place of elements
        }

//Now the array is 2b, 1, 2a in order


        if (freeze_count != 0){
                for (int i = n - freeze_count; i > (n - freeze_count) / 2; i--){

                        if (i != n - i - freeze_count && arr[n - i - freeze_count] < arr[i]) {

                                int temp = arr[n - i - freeze_count];

                                arr[n - i - freeze_count] = arr[i];

                                arr[i] = temp;

                        }

                }

        }

}
```

After the first loop of freeze_count there is no chance that 2a goes to a place that is before 2b. Since the algorithm chances the places of 2a and 2b different than the input elements, the algorithm is not stable and can not be used as intermediate algorithm instead of counting sort.

Now, we need to check that whether Gnome Sort is stable or not by giving the same input array including 1, 2a and 2b elements in order.

```
function gnomeSort(a[0..size-1])

        i := 1

        j := 2

        while i < size do
        if a[i-1] <= a[i] then          // First and second  loop enters here
         i := j                         // No change of any place
        j := j + 1
         else
          swap a[i-1] and a[i]
          i := i – 1
          if i = 0 then
           i := j
           j := j + 1
          endif
        endif
    done
```

By inspecting the algorithm, we can see that input array stays the same. There is no change of any place. If there are elements which are 5a, 5b or 5c, the algorithm just enters the first if and does not change any place of the input array. Therefore, we can say that Gnome Sort Algorithm is a kind of stable algorithm and can be clearly used as an intermediate of Radix Sort Algorithm.

PART B

QUESTION 1

The running time of Heap Sort with respect to different input numbers are given in the table below.

Algorithm / Number of Inputs	1	100	10000	1000000	100000000
Heap Sort	2e - 06	2.2e - 05	0.00342	0.580921	4.5913

Generally, the running time of Heap Sort is measured as O(n* logn). So, the algorithm behaviour with respect to input sizes resembles Merge Sort Algorithm. As seen from the table that, Heap Sort Algorithm run time does not change very much with respect to input sizes, just like Merge Sort. Then, by inspecting the results in the graph above, we can see that Heap Sort Algorithm run time complexity is O(n* logn).

QUESTION 2

I have found the results that:

Total CP's of Clan A is: 50460159

Total CP's of Clan B is: 50528505

And Clan B wins the tournement eventually.