EE 602 – Algorithm I **(UH Manoa, Fall 2020)**

Algorithms Assignment 8

October 24, 2020

Instructor: Yingfei Dong *Name: Frendy Lio Can*

Problem 1

We can do something similar to COUNTING-SORT (A, B, k). We will sort the array so that C[i] contains the number of elements less than or equal to i.
When we are asked how many integers fall into a range[a..b], we can compute C[b] - C[a - 1]. This operation takes O(1) time.

Problem 2

Stable: Insertion and Merge sort.
Not Stable: Heapsort and quicksort.

How to make any sorting algorithm stable:

We can try to make an array of 2D vector where we storage the value and index of the element.

For example, $[1, 1, 3, 2, 4]$ will be $[(1, 0), (1, 1), (3, 2), (2, 3), (4, 4)]$.

After this, we organize our array to follow this constraint $(i, j) < (k, l)$ iff $i < k$ or $(i = k \wedge j < l)$. Where (i, j) represent one 2D vector and (k, l) another 2D vector from the new array.

With this, any algorithm is guaranteed to be stable because each of our elements (first value of our 2D vector) is different. If we have a repeated number, we use the second value of the 2D vector, which is the index, to compare and to ensure if there is a repeated element in the original array it will be also repeated in the new array.

Additional time: Same
Additional Space: Double

Problem 3

Assume that the radix sort works for n = k - 1. Thus, if this sorting algorithm works for k -1; it should also work for k indexes.

Let a_k and b_k be two digits that needs to be sorted.

If $a_k > b_d$ the radix sort places a_d before b_d. If $a_k < b_k$ the radix sort places a_d in front of b_d. If

$a_k = b_d$ the radix sort leaves them in the same place/location. If k digits are the same, this implies that they are sorted for k-1 elements. If the intermediate sort is not stable, then for $a_k = b_k$ we will not have the elements in the same order as the original input.

Thus, by induction, radix sort is true.

Problem 4

We can do divide and conquer algorithm. We will split our array into pairs and compare each pair with each other. After we find the smallest from each pair, we combine them and split them into pairs again. We repeat these until we have 2 elements. We will have the smallest and second smallest element.

When we split the array, the time complexity is $n-1$ and when we do each comparisons, the time complexity is $\lceil (lgn) \rceil - 1$. Thus, at worst case, the time complexity is $n + \lceil (lgn) \rceil - 2$.

Problem 5

Group of 7:

Yes, it will work. This is because we will know the median of medians is less than 4 elements from half of the $\lceil (\frac{n}{7}) \rceil$ groups. Thus, the median of medians is around $\frac{4n}{14}$ of elements. This implies that we are not calling the function recursively on more than $\frac{10n}{14}$ times.

If we assueme that $T(n) < cn$ for $n < k$n the recurrence formula will be the following:

$$\begin{aligned} T(n) =& T(\frac{n}{7}) + T(\frac{10n}{14}) + O(n) \\ T(m) =& T(\frac{m}{7}) + T(\frac{10m}{14}) + O(m), m \geq k \\ \geq& cm(\frac{1}{7} + \frac{10}{14}) + O(m) \end{aligned}$$

With this, as long as we have a constant $c_i \leq \frac{c}{7}$, we will be able to divide into a group of 7 that will work with linear time.

Group of 3:

Following the same deduction for a group of 7, the recurrence formula will be the following:

$$\begin{aligned} T(n) =& T(\frac{n}{3}) + T(\frac{4n}{6}) + O(n) \\ T(m) =& T(\frac{m}{3}) + T(\frac{4m}{6}) + O(m), m \geq k \end{aligned}$$

Since we want to argue that it does not run in linear time, we need to prove that it runs in a different time. Assume that it runs in lg time.

$$T(m) \geq c(\frac{m}{3})lg(\frac{m}{3}) + c(\frac{4m}{6})lg(\frac{4m}{6}) + O(m)$$

Thus, we can observe that it runs greater than lg time. Therefore, a group of 3 will not be linear time.

Problem 6

$$\begin{aligned} X_{k,l} =& I(\text{Collision occurred}), 1 \leq k \leq n \wedge 1 \leq l \leq n \\ =& I(h(k) = h(l) || k \neq l) \\ =& \begin{cases} 1 & , h(k) = h(l) || k \neq l \\ 0 & , \text{otherwise} \end{cases} \end{aligned}$$

Let X be a random variable that counts the number of collisions. Thus, $X = \sum_{k \neq l} X_{k,l}$.

Thus, we can calculate the expectation of the collision.

$$\begin{aligned} E(X) =& E(\sum_{k \neq l} X_{i,j}) \\ =& \sum_{k \neq l} \frac{1}{m} \\ =& \binom{n}{2} \frac{1}{m} \\ =& \frac{n(n-1)}{2m} \end{aligned}$$

Problem 7

```
HASH-DELETE(T, k)
  i = 0
  j = 0
  m = 0
  while(T[j] is not null and i != m)
    j = h(k,i)
    if T[j] == k
      T[j] = DELETE
      return j
    else
      i = i + 1
  return NULL

HASH-INSERT(T, k)
  i = 0
  j = 0
  m = 1
  while(i != m)
    j = h(k,i)
    if T[j] == NULL or T[j] == DELETE
```

```
        T[j] = k
        return j
    else
        i = i + 1
return NULL
```

Problem 8