EE602: ALGORITHMS

HOMEWORK #4

Due: 11/02

Hualiang Li

(1) Ex. 8.2-4

```
Count(arr, a, b)
Pre-process begin:
    Let C[0…n] be a new array
    For i=0 to k
        C[i] = 0
    For j=1 to arr.length
        C[arr[j]] = C[A[j]] + 1
    For i=1 to k
        C[i] = C[i] + C[i-1]
Pre-process end
    If (a>1)
        Return C[b] – C[a-1]
    Else return C[b]
```

(2) Ex. 8.3.2

Stable: Insertion sort, merge sort

Not stable: Heapsort, quicksort

We can make any algorithm stable by mapping the array to an array of pairs, where the first element in each pair is the original element and the second is its index. Then we sort lexicographically. This scheme takes additional $\Theta(n)$ space.

(3) Ex. 8.3.3

Initialization. The array is trivially sorted on the last 0 digits.

Maintenance. Let's assume that the array is sorted on the last i-1 digits. After we sort on the ith digit, the array will be sorted on the last i digits. It is obvious that elements with different digit in the ith position are ordered accordingly; in the case of the same ith digit, we still get a correct order, because we're using a stable sort and the elements were already sorted on the last i-1digits.

Termination. The loop terminates when i=d+1. Since the invariant holds, we have the numbers sorted on d digits.

(4) Ex. 9.1.1

We can compare the elements in a tournament fashion - we split them into pairs, compare each pair and then proceed to compare the winners in the same fashion. We need to keep track of each "match" the potential winners have participated in.

We select a winner in n−1 matches. At this point, we know that the second smallest element is one of the lgn elements that lost to the smallest - each of them is smaller than the ones it has been compared to, prior to losing. In another ceil(lgn)−1 comparisons we can find the smallest element out of those. So, the total comparison is n-1 + ceil(lgn) -1 = n+ceil(lgn)-2.

(5) Ex. 9.3-1

Groups of 7

The algorithm will work if the elements are divided in groups of 7. On each partitioning, the minimum number of elements that are less than (or greater than) x will be:

$$4(0.5*n/7-2) >= 2n/7 - 8$$

The partitioning will reduce the subproblem to size at most $5n/7+8$. This yields the following recurrence:

$$T(n) = T(n/7) + T(5*n/7 + 8) + O(n) \text{ if } n >= n_0$$

$$T(n) = O(1) \text{ otherwise}$$

We guess $T(n) \leqslant cn$ and bound the non-recursive term with an:a

$$\begin{aligned}
T(n) &<= c(n/7)+c(5n/7+8) + an \\
&<= cn/7 + c + 5cn/7 + 8c + an \\
&= 6cn/7 + 9c + an \\
&= cn+(-cn/7+9c+an) \\
&<= cn \\
&= O(n)
\end{aligned}$$

The last step holds when $(-cn/7+9c+an) \leqslant 0$. That is:

$$\begin{aligned}
&-cn/7 + 9c + an <= 0 \\
&c(n/7-9) >= an \\
&c(n-63)/7 >= an \\
&c >= 7an/(n-63)
\end{aligned}$$

If we pick $n_0 = 126$ and $n<=n_0$, we get that $n/(n-63) <= 2$. Ten we just need $c>= 14a$.

Groups of 3

The algorithm will not work for groups of three. The number of elements that are less than (or greater than) the median-of-medians is:

$$2(0.5*n/3-2) >= n/3-4$$

The recurrence is thus:

$$T(n) = T(n/3) + T(2n/3+4)+O(n)$$

We're going to prove that $T(n)=\omega(n)$ using the substitution method. We guess that $T(n)>cn$ and bound the non-recursive term with an.

$$\begin{aligned}
T(n) &> c(n/3) + c(2n/3+2) + an \\
&> cn/3+c+2cn/3+2c+an \\
&= cn+3c+an \\
&> cn
\end{aligned}$$

Therefor, it does not run in linear time.

(6) Ex. 11.2-1

$$\binom{n}{2} * \frac{1}{m} = (n*(n-1)/2)/m$$

(7) Ex. 11.4-2 HASH-DELETE

```
HASH-DELETE(T, k):
    i <- 0
    repeat j <- h(k, i)
        if T[j] == k:
            T[j] = "DELETED"
            return
        else if T[j] == NIL:
            break
        else:
            i <- i + 1
    until i == m
    error "k is not in T"


HASH-INSERT(T, k):
    i <- 0
    repeat j <- h(k, i)
        if T[j] == NIL || T[j] == "DELETED":
            T[j] = k
            rteurn j
        else i <- i + 1
    until i == m
    error "hash table overflow"
```

(8) Problem 11.1

(a) Inserting a key entails an unsuccessful search followed by placing the key into the first empty slot found. if we let X be the random variable denoting the number of probes in an unsuc- cessful search, then Pr{X $\geqslant$ i} $\leqslant$ αi−1. Since n $\leqslant$ m/2, we have α $\leqslant$ 1/2. Let- tingi =k+1,wehavePr{X >k}=Pr{X $\geqslant$k+1}$\leqslant$(1/2)(k+1)−1 =2−k.

(b) Substituting k = 2 lg n into the statement of part (a) yields that the probability that the ith insertion requires more than k=2lgn probes is at most $2^{-2\lg n}$ = $(2\lg n)^{-2}$ =n^{-2} =$1/n^2$.

(c) Let the event A be X > 2lgn, and for i = 1,2,...,n, let the event Ai be Xi > 2lgn. In part (b), we showed that Pr{Ai} $\leqslant$ 1/n2 for i = 1,2,...,n. From how we defined these events, A = A1 $\cup$ A2 $\cup$ $\cdots$ $\cup$ An. We have

Pr{A} $\leqslant$ Pr{A1}+Pr{A2}+ $\bullet\bullet\bullet$ +Pr{An} $\leqslant$ n/n^2 = 1/n.

d. We use the definition of expectation and break the sum into two parts:

$$E[X] = \sum_{k=1}^{n} k \cdot \Pr\{X = k\}$$

$$= \sum_{k=1}^{\lceil 2\lg n \rceil} k \cdot \Pr\{X = k\} + \sum_{k=\lceil 2\lg n \rceil+1}^{n} k \cdot \Pr\{X = k\}$$

$\leqslant \sum_{k=1}^{\lceil 2\lg n \rceil} \lceil 2\lg n \rceil \cdot \Pr\{X = k\} + \sum_{k=\lceil 2\lg n \rceil+1}^{n} n \cdot \Pr\{X = k\}$

$= \lceil 2\lg n \rceil \sum_{k=1}^{\lceil 2\lg n \rceil} \Pr\{X = k\} + n \sum_{k=\lceil 2\lg n \rceil+1}^{n} \Pr\{X = k\}$

Since X takes on exactly one value, we have that $\sum_{k=1}^{\lceil 2\lg n \rceil} \Pr\{X = k\} = \Pr\{X \leqslant \lceil 2\lg n \rceil\} \leqslant 1$ and $\sum_{k=\lceil 2lgn \rceil+1}^{n} \Pr\{X = k\} \leqslant \Pr\{X > 2\lg n\} \leqslant 1/n$, by part (c). Therefore, $E[X] \leqslant \lceil 2\lg n \rceil \cdot 1 + n \cdot (1/n) = \lceil 2\lg n \rceil + 1 = O(\lg n)$.