Q4:-1

99

SULOCHANAVANAMALAI
U4128-1114

Introduction to Theory of Algorithms, Midterm II. This is a closed book, closed notes exam and no calculators are allowed. Oct. 26th, 2016

Question 1 (30 points): Describe the *Quicksort* algorithm. You should also explain how the partition function works.

⇒ Quick sort algorithm sorts in place. It also follows divide and conquer algorithm method.

Algorithm:

```
QuickSort(A,p,r)
{ if (p<r)
  { q = Partition(A,p,r);
    Quicksort(A,p,q-1);
    QuickSort(A,q+1,r);
}

Partition(A,p,r)
  x <- A[r]
  i <- p-1
  for j <- p to r-1
  if A[j] ≤ x
    i <- i+1
    exchange A[i] <-> A[j]
  exchange A[i+1] <-> A[r]
  return i+1
```

Partition algorithm in quicksort considers a pivot element as 'x' and when the for loop begins, if the element (j) is less than pivot (x) element; then $A[i] \leftrightarrow A[j]$; if the element $A[j]$ is greater than pivot; then it increments and follows the first case. ~~Pivot~~ Once the for loop ends, the pivot element is placed in position where the elements are ~~A[p..~~ (A,p,q-1)....are less than pivot while (q+1...r) are greater than pivot.

Worst case: $O(n^2)$

Average case: $O(n\log n)$

Best case: $O(n\log n)$

$A[p \ldots r] \longrightarrow (p \ldots q-1)\,\underset{\text{pivot}}{[\,]}\,(q+1 \ldots r)$ ← Array gets sorted in place. It doesnot need additional one to sort

Question 2 (20 points): Prove that any comparison sorting algorithm that sorts *n* distinct keys requires at least lg *n*! comparisons in the worst case.

Consider the comparision algorithm as Decision tree. Decision tree ~~is almost a~~ represents a binary tree having leaf nodes indicated in sorted order. If there are 'n' distinct elements, then No. of leaves for size n happens in n! permutations for each leaf in decision tree.

(i) Consider height of binary tree: $h \le 2^h$.

$i \le 2^h$

(ii) $I = n!$ & $I_{max} = 2^h \Rightarrow n! \le 2^h$; hence $\log(n!) \le h$; after apply log on both sides

According to Stirling's approximation; $n! > \left(\frac{n}{e}\right)^n$; $\Rightarrow h \ge \lg(n!)$

$$\Rightarrow h \ge \log\left(\frac{n}{e}\right)^n \Rightarrow n\log n - n\lg e.$$

$$\Rightarrow h = \Omega(n\log n)$$

Hence, when comparision algorithm ~~alg~~ above considers $n! \le 2^h$, when log is applied on both sides, and after using Stirlings approximation, it takes log n! comparisions in worst case leading to $\Omega(n\log n)$ time complexity for n distinct elements

Question 3 (10 points) Answer the following questions about the bucket sort algorithm.

(a) (5 points) What's the assumption of the input of the bucket sort algorithm?

The assumption of the input of bucket sort algorithm takes $[0,1)$ elements distributed uniformly. Input is generated by random process. 'n' equal sized buckets merely a linked list is divided into subintervals of $1/n$ size. And each element is sorted in bucket sort using InsertionSort.

(b) (5 points) Prove that the expected number of element in each bucket is 1

Consider the Time complexity for BucketSort;

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

when Expected numbers of element in each bucket is '1', i.e., uniform distribution of elements takes place in linear time: $O(n)$.

$$E\left(T(n)\right) = E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \Rightarrow \Theta(n) + \sum_{i=0}^{n-1} O\left(E[n_i^2]\right)$$

For each bucket; $E\left[X_{ij}^2\right] = 1/n$

$$E[n_i^2] \neq E\left[X_{ij} \cdot X_{ik}\right] = \frac{1}{n^2} \quad \left[\because \frac{1}{n} \times \frac{1}{n} = \frac{1}{n^2}\right]$$

$$\Rightarrow \cancel{T(n)} \; E\left[n_i^2\right] = \sum_{j=1}^{n} \frac{1}{n} + 2\sum_{j=1}^{n-1} \sum_{k=j+1}^{n} \frac{1}{n^2} = 1 + 1 - \frac{1}{n} = 2 - \frac{1}{n}$$

claim: $n_{ij} = \sum_{j=1}^{n} x_{ij}$; Hence

$$E\left[T(n)\right] = \Theta(n) + \sum_{i=0}^{n-1} O\left(2 - 1/n\right) \Rightarrow \Theta(n) + O(n) = O(n)$$

Each bucket when given uniformly distributed, linear time will be $O(n)$. ~~From the time~~, When each bucket gets atleast one element in $O(n)$ time, the expected number of element in each bucket is 1.

SULOCHANAVANAMALAI

-1

Question 4 (10 points): Array A has n elements and each element is a non-zero real number. Design an algorithm to rearrange the order of the elements in A, so that the negative numbers appear before the positive numbers. For example,

Before sorting: A = [-1, 1, 7, -11, 6, -9]. After sorting, A= [-1, -11, -9, 1, 7, 6].

Your algorithm should have the O (n) worst case time complexity and it should be in place (e.g., no allocation of additional arrays). Note that you don't have to maintain the original order of the negative number or positive numbers in the sorting result. For example, A= [-11, -1, -9, 6, 7, 1] is also a correct output.

Sort (A, ~~p, r~~ n)

The algorithm runs in O(n)

```
{
  x <- 0
  i = ~~p~~ -1
  for j <- ~~p~~ 0 to n-1
     if (A[j] <= x)
     i <- i+1
  Exchange A[i] <-> A[j]
     return A
}
```

Question 5 (10 points): Is it possible to sort n integers in the range 0 to $n^{n-1} - 1$ in O(n) time. If no, please explain your reason. If yes, please write down your algorithm.

⇒ No, it cannot be sort in $O(n)$.

Proof: Given n integers; from range 0 to $n^{n-1}-1$; $T(n) = O(d(n+k))$

$d \Rightarrow$ no. of digits $\rightarrow d = \log_k n$; k is the no. of comparisions made.

$d = \log_k (n^{n-1} - 1)$

$d = \log_k n^{n-1}$ (if k=n)

$d = (n-1)$

then $T(n) = O((n-1)(n+n))$

$= \theta((n-1)(2n))$

$= 2n^2 - 2n$

$\leq O(n^2)$

Hence, the time taken to sort n integers ranging from 0 to $n^{n-1}-1 \Rightarrow O(n^2)$.

Question 6 (20 points): The following algorithm, called the big five algorithm, both because of the cardinality and stature of its inventors (Blum, Floyd, Pratt, Rivest, and Tarjan) and because of the importance of the number 5 in its design, returns the k-th smallest element of the input set S.

function BIGFIVE(S, k)
  Put the elements of S into groups of 5 and sort each group
  Let M = {y | y is the 3rd smallest key (median) in its group}
  Let x = BIGFIVE (M, $\lfloor |M|/2 \rfloor$)
  Use x as a pivot to divide S − {x} into two subsets
    L = {y ∈ S − {x} | y ≤ x}
    U = {y ∈ S − {x} | y > x}
  if k == |L|, **then return x**
  else if k < |L|, **then return** BIGGIVE(L, k)
  else return BIGGIVE(U, k-|L|-1) **end if**
end BIGFIVE

Answer the following questions.

(a) (4 points) Is x the median of Set S? Explain your reason.

~~No,~~ ~~Yes it is not~~ No, it is ~~not~~ median of set S.

Once the elements are divided in terms of 5 groups; they are again sorted and the middle element is the 3rd element. forms the lower median.

(b) (8 points) Prove that the worst case running time of the BIGFIVE algorithm is O($nlgn$) if we put the elements of S into groups of 3. Assume that n is a multiple of 3.

If the algorithm divides the elements of S in group of 3; then;
The numbers greater than the 'x', are to atleast having 2 elements:

then; at $T(\lceil\frac{n}{3}\rceil) = 2\left[\left(\frac{1}{2}\times\frac{n}{3}\right)-2\right] \Rightarrow \frac{n}{3}-4$

at the elements from $n \rightarrow n-\left(\frac{n}{3}-4\right) = \frac{2n}{3}+4$

$$T(n) = T\left[\frac{n}{3}\right] + T\left[\frac{2n}{3}+4\right] + O(n)$$

$$\le \frac{cn}{3} + cn\left[\frac{2n}{3}+4\right] + cn$$

$$\le \frac{cn}{3} + \frac{2cn^2}{3} + 4c + cn$$ $\Rightarrow$ It doesnot run in linear time and assume $\therefore n = lgn$ elements

$\Rightarrow 2c[\log n+1] \approx \log n$ for an element $\Rightarrow T(n) = \Omega(n\log n)$ for groups of 3 //.

The size of sublist is not reduced to n elements, it is greater.

(c) (8 points) Prove that the worst case running time of the BIGFIVE algorithm is O(n) if we put the elements of S into groups of 7.

If partitions runs if we put the elements of S into groups of 7, then half of the sublists have 4 elements greater than pivot, i.e.,

$$T(n) \le T\left(\lceil\frac{n}{7}\rceil\right) + T\left(\frac{5n}{7}+8\right) + O(n)$$

where i.e., when elements greater than pivot; then $4\left[\left(\frac{1}{2}\times\lceil\frac{n}{7}\rceil\right)-2\right] \ge \frac{2n}{7}-8$

For the rest of the elements $\Rightarrow n - \left(\frac{2n}{7}-8\right) = \frac{5n}{7}+8$

$\because O(n) = cn$

Hence; $$T(n) = T\left(\lceil\frac{n}{7}\rceil\right) + T\left(\frac{5n}{7}+8\right) + O(n)$$

$$= c\left[\frac{n}{7}\right] + c\left(\frac{5n}{7}+8\right) + cn$$

$$= \frac{cn}{7} + c + \frac{5cn}{7} + 8c + cn$$

$$= \frac{6cn}{7} + cn + 9c \Rightarrow \frac{13cn}{7} + 9c$$

[eliminate constants]

$$= cn\cdot\left(\frac{13}{7}\right) + 9c$$

Thus; it runs in $O(n)$ in worst case. $\approx cn \quad \therefore T(n) = O(n)$ //.

Question 7 (20 bonus points): How do you modify Quicksort so that the worst case running time of Quicksort is $O(n\lg n)$? Please give the pseudocode of your implementation and analyze the running time.

In worst case, Quicksort produces bad split, that is having 0 and n-1 elements.
(or) if all the elements are in reverse order.

1) Partition: $(0:n-1,\ 1:n-2,\ 2:n-3,\ \cdots\ n-2:1,\ n-1:0)$ with probability of $1/n$ for each element.

$$T(n) = \frac{1}{n}\sum_{K=0}^{K=n-1} T(K) + T(n-1-K) + \Theta(n)$$

$$= \frac{1}{n}\sum_{K=0}^{K=n-1} 2\left(T(K) + \Theta(n)\right) \qquad \left[\because T(0)+T(n-1)+T(1)+T(n-2)+\cdots T(n-1)+T(0)\right].$$

$$= \frac{2}{n}\sum_{K=0}^{K=n-1} T(K) + \Theta(n) \Rightarrow \text{Assume } T(n) = O(n\log n) \Rightarrow T(K) \le aK\lg K + b \quad a>0;\ b>0$$

then $T(n) = \frac{2}{n}\sum_{K=0}^{n-1}(aK\log K + b) + \Theta(n)$

at $n=0$; $\lim_{n\to 0} an\lg n = 0$, hence

we consider,

$$T(n) \le \frac{2}{n}\left[b + \sum_{K=1}^{n-1}(aK\lg K + b)\right] + \Theta(n) \Rightarrow \frac{2a}{n}\sum_{K=1}^{n-1} K\lg K + \frac{2b}{n}(n-1) + \Theta(n)$$

$$\Rightarrow \sum_{K=1}^{n-1} K\lg K \le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$$

$$T(n) \le \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + 2b + \Theta(n) \le \qquad \because n-1<n;\ \frac{2b(n-1)}{n} < 2b$$

$$\le an\lg n - \frac{an}{4} + 2b + \Theta(n) \Rightarrow \le an\lg n + b$$

eradicating constants $\rightarrow T(n) = O(n\lg n)$

2) when the split happens in 9:1; then

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn$$

shortest path $= \left(\frac{1}{10}\right)^i cn \rightarrow i = \log_{10} cn = (\log n)$

longest path $= \left(\frac{9}{10}\right)^i cn \Rightarrow i = \log_{10/9} cn = O(\log n)$

$T(n) \ge (1+\log_{10} n)\,cn = \Omega(n\log n)$

$T(n) < (1+\log_{10/9} n)\,cn = O(n\log n)$

$\Rightarrow \Theta(n\log n) = T(n)$

Pseudocode:

```
Quicksort (A,p,r)
{ if (p<r)
  {
   q = Partition (A,p,r);
  Quicksort (A,p,q-1);
  Quicksort (A,q+1,r);
  }
```

```
Partition(A,p,r)
  x <- A[r]
  i <- p-1
 for j <- p to r-1
 if A[j] <= x
    i <- i+1
 exchange A[i] <-> A[j]
 exchange A[i+1] <-> A[r]
 return i+1;
```

Questions 4 (10 points): Given the RADIX-SORT below, why do we need to use a stable sort for sorting each digit i?

RADIX-SORT(A, d)

for i= 1 **to** d

StableSort(A) on digit i

→ Radix sort is basically sorts the elements by each digit. So, if we use non-stable sort over an array of digits, when we call next time if we have some digits we can have wrong sorting.

For exemple:

Non stable

123
124
356 → $\boxed{124}$ $\boxed{123}$ $\boxed{356}$

Stable

$\boxed{123}$ $\boxed{124}$ $\boxed{356}$

↳ In this exemple, after sorting $\boxed{4}$ and $\boxed{3}$ then we have some digits $\boxed{2}$. If we mixed the order then the result become incorrect

Questions 5 (10 points): Prove that any comparison sorting algorithm that sorts n distinct keys requires at least lg n! comparisons in the worst case.

| 3 | 5 | 1 | 2 | 4 | 7 | 6 |

In decision tree

we have min # of leaves ⇒ $n!$

" " max # of leaves ⇒ 2^h

→ So to compare:

$2^h > n!$

$\lg 2^h > \lg n!$

$h > \lg n!$ → So, in any comparison algorithm, we need at least $\lg n!$

B/D 5. Which of the following is CORRECT?

(A) The worst-case running time of Randomized Quicksort is $\Theta(nlgn)$ ×

(B) It is possible to improve the worst case running time of Quicksort from $\Theta(n^2)$ to $\Theta(nlgn)$ ✓

(C) To sort an n-element array, the running time of counting sort is O(n) × assume k is large

(D) None of the above

Answer: D ~~A~~ ✓

C/D 6. Which of the following can be sorted by Bucket Sort with a $\Theta(n)$ running time?

(A) An array whose elements are probabilities — probabilities can be 1.

(B) An array whose elements are binary bits

(C) An array whose elements are birthdays (assume 365 days per year) ×

(D) An array whose elements are random fractional numbers ✓

Answer: D ~~A~~ ✓

7. Given a set of n integers in the range $[1,\ n^3]$, which of the following algorithm can achieve a running time of $\Theta(n)$?

(A) Counting sort

(B) Radix Sort

(C) Bucket Sort (assume that the n integers can be mapped to n fractional numbers that falls in the interval [0,1))

(D) None of the above

Answer: B ✓

C 8. Which of the following it CORRECT?

(A) Counting sort is stable and "in place" ×

(B) In quicksort, each pair of elements are compared more than once ×

(C) The dominant cost of Quicksort is Partitioning ✓ call O(n) time don't mean take O(n) time

(D) All of the above

Answer: B ×

A 9. Suppose the partition function results in an 86-to-14 split. What's the running time of Randomized-Select()?

(A) O(n)
(B) $O(n^2)$
(C) O(nlgn)
(D) None of the above

Answer: C ✗

10. What is the smallest number of comparisons in order to find both minimum and maximum of an n-element array?

(A) O (2n/3)
(B) O (n-1)
(C) O (2(n-1))
(D) O(3n/2)

Answer: D

Questions 2 (2 points for each question): Use appropriate answers to fill in the following blanks

-2

1. To sort sequence {2, 5, 1, 7, 3, 4, 0}, the counter array C generated by counting sort is {1, 2, 3, 4, 5, 6, 6, 7} ✓

2. Any comparison sort algorithm requires at least O(nlgn) comparisons in the worst case ✓

3. The minimum number of leaves on a decision tree is n! ✓, and the maximum number of leaves on a decision tree is 2^h ✓

4. The worst-case time for Quicksort on an input of size n can be represented by the recurrence T(n) = max(T(n-1) + T(0) + Θ(n)) ✗

5. For bucket sort, let n_i be the number of elements placed in bucket B[i]. The running time of bucket sort can be expressed as T(n) = $\Theta(n) + \sum_{i=0}^{n-1} (n_i^2)$

Heapsort :　　$\lfloor 7/2 \rfloor$ downto 1

Build-MaxHeap(A) // 6 times. = 2+2+2

A[heap-size] = ExtractMax(A) // 4+2+3+2+1

A.heap-size = A.heap-size − 1　∴ Total = 6+12 = 18

Max-Heapify(A, 1).

$T(n) = T(n-1) + n$.

b. (5 points): Quicksort (not randomized) on 7 distinct items that are already correctly sorted in descending order.

Solve:　$n=7$.　Pivot: ~~A[r]~~

A[r]　　$n-1 + n-2 + \cdots + 1$

$= \frac{n(n-1)}{2} = \frac{7\times 6}{2} = 21$

7 6 5 4 3 2 1

1 7 6 5 4 3 2

1 2 7 6 5 4 3

Questions 4 (10 points): We have an input array A with n ($n\geq 2$) numbers.

a. (5 points): Describe a O(n) worst-case time algorithm to find two elements $x, y \in A$ such that $|x-y| \geq |u-v|$ for all $u, v \in A$

Solve:　$|x-y| \geq |u-v|$.　~~Counting~~ Sort — O(n).

Step1: Calculate the distance between every two element.

Step2: Sort all the distances. [Counting Sort corresponding] → O(n).

Step3: Return the biggest distance. x, y. satisfy the definition

$C_n^2 = \frac{n(n-1)}{2}$

b. (5 points): Describe a O($n$lg$n$) worst-case time algorithm to find two elements $x, y \in A$ such that $|x-y| \leq |u-v|$ for all $u, v \in A$

Solve: Use Merge-Sort to sort the distance of any two elements.

then select the least one, i.e. x, y.

can satisfy that $|x-y| \leq |u-v|$ for all $u, v \in A$.

Merge-Sort(A). → O(nlgn). worst-case. running time

Questions 1 – 10. Multiple choices. (Total 50 points. 5 points/question.)
Choose appropriate answer and write it on the blank below the question. (ONLY ONE answer to each question)

-30 ×10

1. What's the running time of Quicksort when the array contains distinct elements and is sorted in decreasing order?

(A) $\Theta(n^2)$
(B) $\Theta(n)$
(C) $\Theta(lgn)$
(D) None of the above

Answer: A ✓

2. To partition an already sorted array with n elements, what's the running time of PARTITION?

(A) $\Theta(n)$
(B) $\Theta(lgn)$
(C) $\Theta(n^2)$
(D) None of the above

Answer: A ✓

D 3. What's the partition result of {2, 8, 7, 1, 3, 5, 6, 4}? (Assume that the last element is the pivot)

text 172

(A) {2, 1, 3, 4, 8, 7, 5, 6}
(B) {3, 2, 1, 4, 8, 7, 5, 6}
(C) {2, 1, 3, 4, 7, 8, 5, 6}
(D) None of the above

Answer: A ✗

D 4. Which of the following is NOT the upper bound of $\sum_{k=1}^{k=n-1} klgk$ (n > 2)?

$\leq \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2$

(A) $n^2\sqrt{n}$
(B) $\frac{1}{2}n^3$
(C) $n^2\lg n - n$
(D) None of the above

Answer: C ✗

Questions 6 (10 points): Sho... time.

$n=3$ | 0 | 3 | 1 | 4 | 3 |

→ Basically we can write to insertion sort that is going t... And we don't need any extra memory.

```
Sort (A,n) {
  for (i=0; i<n-1; i++)
    for (j=i; j<n; j++)
      if (A[i] >= A[j]) {
        swap A[i] with A[j]
  }
```

runs n times

runs n times

Total ⟹ $\underline{\underline{O(n^2)}}$

Questions 7 (10 points): If there are n keys and the hash table has m slots, what's the probability that at least one slot contains at least k keys? Please show your steps. Assuming simple uniform hashing and the chaining strategy for collision resolution.

n keys

m

Hash (k) = k mod m

Failed search: $O(1+\alpha)$

Success search: $O\left(1+\frac{\alpha}{2}+\frac{\alpha}{2n}\right)$

$Pr(x_{ij})$ → probability of key in slot m.

$Pr(x_{ij}) = \sum_{i=0}^{m} \sum_{j=i}^{n} \frac{n-i-j}{m-i-j}$

-10

$\frac{m}{n} = \alpha$ → so the probability of one slot has at least k keys will be α.

Then, when we make search on hash table it will cost us $O(1+\alpha)$

Questions 6 *(10 points): Show how to sort n integers in the range 0 to* $n^n - 1$ in $O(n^2)$ *time.*

$n = 3$ [0|3|1|4|5|2|8|8|2]

→ Basically we can write an algorithm similar to insertion sort that is going to sort in $O(n^2)$ and we don't need extra memory.

```
sort(A,n){
  for(i=0; i<n-1; i++)
    for(j=i; j<n; j++)
      if(A[i] >= A[j]){
        swap A[i] with A[j]
  }
```

runs n times

runs n times

Total ⟹ $O(n^2)$

Questions 7 *(10 points): If there are n keys and the hash table has m slots, what's the probability that at least one slot contains at least k keys? Please show your steps. Assuming simple uniform hashing and the chaining strategy for collision resolution.*

n keys

Hash(k) = k mod m

Failed search	Success search
$O(1+\alpha)$	$O\left(1+\frac{\alpha}{2}+\frac{\alpha}{2n}\right)$

$Pr(x_{ij})$ → probability of key in slot m.

$Pr(x_{ij}) = \sum_{i=0}^{m}\sum_{j=i}^{n} \frac{n-i-j}{m-i-j}$

-10

$\frac{m}{n} = \alpha$ → so the probability of one slot has at least k keys will be α.

(c) (8 points) Prove that the worst case running time of the BIGFIVE algorithm is O(n).

In each group, at least used $3(\frac{1}{2}\lceil\frac{n}{5}\rceil - 2) \geq (\frac{3n}{10} - 6)$ elements.

so after one split, There are at most $\frac{7n}{10} + 6$ elements.

So the total cost can be presented as

$T(n) = T(\lceil\frac{n}{5}\rceil) + T(\frac{7n}{10} + 6) + \Theta(n)$

we guess $T(n) = O(n)$, $T(n) \leq cn$

$$T(n) \leq \frac{1}{5} \cdot cn + c \cdot (\frac{7n}{10} + 6) + an.$$
$$= \frac{cn}{5} + \frac{7}{10} cn + 6c + an$$
$$= \frac{9}{10} cn + 6c + an$$
$$= cn - \frac{1}{10} cn + 6c + an$$

to let $6c + an - \frac{1}{10} cn \geq 0$ we just need to find an a and c to make $a - \frac{1}{10} c \geq 0$

because we can find such a and c, so $T(n) \leq cn$.

So BIGFIVE algorithm's worst case is $O(n)$.

Questions 5 (15 points): The following algorithm returns the k-th smallest element of the input set S. It is called the big five algorithm, both because of the cardinality and stature of its inventors (Blum, Floyd, Pratt, Rivest, and Tarjan) and because of the importance of the number 5 in its design. Answer the following questions.

function BIGFIVE(S, k)
  Put the elements of S into groups of 5 and sort each group
  Let M = {y | y is the 3rd smallest key (median) in its group}
  Let x = BIGFIVE (M, $\lfloor |M|/2 \rfloor$)
  Use x as a pivot to divide S − {x} into two subsets
    L = {y ∈ S − {x} | y ≤ x}
    U = {y ∈ S − {x} | y > x}
  if k == |L|, **then return x**
  else if k < |L|, **then return** BIGGIVE(L, k)
  else return BIGGIVE(U, k-|L|-1) **end if**
end BIGFIVE

(a) (2 points) In line 4, x = BIGFIVE (M,$\lfloor |M|/2 \rfloor$). What's the return value of BIGFIVE (M,$\lfloor |M|/2 \rfloor$)?

M contains all the median in each group.
there are $\lceil \frac{S}{5} \rceil$ groups.
$|M|$ is the number of elements in M and it is equal to $\lceil \frac{S}{5} \rceil$
So the BIGFIVE (M, $\lfloor |M|/2 \rfloor$) return the $\lfloor |M|/2 \rfloor$ smallest element in M.
or the median

(b) (5 points) Is the return value of BIGFIVE (M,$\lfloor |M|/2 \rfloor$) equal to the median of S? Explain your reason.

no. BIGFIVE (M, $\lfloor |M|/2 \rfloor$) just find the median in M.
But this median may not be the median of S.
For example, for set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.
put to two groups {1, 2, 3, 4, 5} {6, 7, 8, 9, 10}.
so M should be {3, 8}.
And x is 3, but the median of S is 5.

(Reminder: question 5-(c) is on the reverse side)

(b) (3 points) Describe the bucket sort algorithm

Create 10 buckets from 0 to 9.
put the numbers into buckets respectly.
in each bucket, sorting the elements by insertion sort.
output the numbers from bucket 0 to bucket 9 to get the sorted array.

(c) (10 points) Prove that the expected number of elements in each bucket is 1

There are n elements

Each element can be put into one bucket

the probability is $\frac{1}{n}$ because it is equal chance to put into

So For each bucket the expected number of elements is $n \times \frac{1}{n} = 1$.

$$X_{ij} = \begin{cases} 1 & \text{if } A[j] \text{ fall in Bucket } i. \\ 0 & \text{if } A[j] \text{ doesn't fall in Bucket } i. \end{cases}$$

$$n_i = \sum_{j=1}^{n} X_{ij}, \quad E(n_i) = E\left(\sum_{j=1}^{n} X_{ij}\right) = \sum_{j=1}^{n} E(X_{ij}) = \sum_{j=1}^{n} P(A[j] \text{ fall in}$$

Questions 3 (10 points): Describe the *Quicksort* algorithm (2 points) and fill in the following blanks to finish the PARTITION function (1 point for each blank).

Quicksort first choose the last element A[r] as pivot. Then compare each element with the pivot. put all elements smaller than the pivot in the front of the Array and all elements bigger than the pivot behind the pivot. And recursively do this until the two parts around the pivot become 1 element. Then we get the sorted array by the quicksort.

PARTITION(A, p, r)

x = A[r]

i = p-1

for j = p to r-1

if A[j] ≤ x

i = i+1;

exchange A[i] and A[j];

exchange A[i+1] and A[r];

return i+1;

Question 4 (15 points) Answer the following questions about the bucket sort algorithm.

(a) (2 points) What's the assumption of the input of the bucket sort algorithm?

-1

the assumption is all the elements we need to sort is in [0,1) uniform

Uniform random distribution

Questions 1 (10 points): What's the input assumption of bucket [...]

The input of the array should be uniformly ranged over $[0, 1)$.

Questions 2 (10 points): What's the worst case running time of Quicksort when the array partition function always results in a 99-to-1split? What's the worst case running time of Quicksort when the array is already sorted? Please give the recurrences.

99 to 1 split $\rightarrow$ $T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right) + \Theta(n) \rightarrow O(n \lg n)$

sorted $\rightarrow$ $T(n) = T(n-1) + \Theta(n) \rightarrow O(n^2)$

Questions 3 (10 points): Describe the *Quicksort* algorithm. You should also explain how the partition function works.

Quicksort is a recursive algorithm with partitioning. Partition first selects a pivot, then splits the array based on this pivot. (Left side the elements are less then pivot, right side the elements are greater then pivot).

Then recursivly calls quicksort over this splited arrays.

→ Quick sort is a inplace algorithm, when it is spliting the array based on pivot it exchanges the elements. It doesn't need an extra memory.

Question 4 (10 points) Show that the running time of Quicksort (not the randomized version) is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

$T(n) = T(n-1) + T(0) + \Theta(n)$

$= \Theta(n^2).$

Question 5 (10 points) Explain why any comparison sort algorithm requires $\Omega(nlgn)$ comparisons in the worst case.

$I_{min} = n!$

$I_{max} \leq 2^h$

$n! \leq 2^h.$

$\log(n!) \leq h$

$n! > (\frac{n}{e})^n$

$\therefore h \geq n\lg n - n\lg e$

$= \Omega(n\lg n)$

By proving that the height of the decision tree

Stirling's approximation

Question 7 (10 points) Given an array A of n elements. Write an $\Theta(n)$ algorithm to find the minimum, maximum, and the i-th largest elements from A.

Question 8 (10 points) Write an algorithm to find the 'next' node (i.e., in-order successor) of a given node in a binary search tree where each node has a link to its parent.

Question 4 (10 points) Show that the running time of Quicksort (not the randomized version) is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

Question 5 (10 points) Explain why any comparison sort algorithm requires $\Omega(nlgn)$ comparisons in the worst case.

min # of leaves ~~in decisi.~~ $n!$

max # of leavage $\longrightarrow 2^h$

$2^h > n!$

$h.\lg 2 > \lg n!$

$h > \lg n! \longrightarrow$

$\leq \lg\left(\frac{n}{e}\right)^n$

$= n\lg n - n\lg e \quad \leq n\lg n$

Question 6 (10 points) Read the algorithm of counting sort below, and answer the following questions

```
COUNTING-SORT(A, B, k)

1 let C[0 .. k] be a new array
2 for i = 0 to k
3     C[i] = 0
4 for j = 1 to A.length
5     C[A[j]] = C[A[j]]+1
6 for i = 1 to k
7     C[i] = C[i] + C[i-1]
8 for j = A.length down to 1
9     B[C[A[j]]] = A[j]
10    C[A[j]] = C[A[j]]-1
```

a. (7 points) Suppose that we were to rewrite the for loop header in line 8 of the COUNTING-SORT as **for** $j = 1$ **to** A.*length*. Show that the algorithm still works properly. Is the modified algorithm stable?

b. (3 points) Is it a good idea to use counting sort to sort a set of n integers in the range of $[1,\ n^3]$? Please explain your answer.