Cpt S 350 Mid-term

Name (PRINT): Hongqi Guo

ID: 011552159

50 minutes.

1. (15 pts, standard) StupidSort is a sorting algorithm that I invented last night. It can sort an array of n integers in worst case time $O(n^3)$. We also know that partition is a procedure, with return value r, that reorganizes an array of n integers and splits the reorganized array into a low-part and a high-part, separated by the element indexed by r. Let mixsort$(A, 1, n)$ be an algorithm that sorts an array A with n integers. It works as follows:

mixsort(A, p, q){
if $p \geq q$, return;
r=partition(A, p, q);
//run mixsort on the low part
mixsort$(A, p, r-1)$;
//run StupidSort on the high part
StupidSort$(A, r+1, q)$;
}

Compute the worst-case time complexity of mixsort. Please faithfully follow the steps of "write a formula", "guess a solution" (Try to guess $O(n^3)$), and "check your solution" and show your math work.

step 1. worst a formula.

$$T_w(n) = \max_{1 \le r \le n} \{ T_w(r-1) + a(n-r)^2 + an^3 \}$$

Step 2. guess $T_w(n) = O(n^3) \le Cn^3$.

step 3. check.

$$T_w(n) = \max_{1 \le r \le n} \{ O(r-1)^2 + O(n-r)^2 + O(n^3) \}$$

$$= \max_{1 \le r \le n} \{ \underbrace{a(r-1)^2 + a(n-r)^2 + an^3}_{F(r)} \}$$

$$F'(r) = 2a(r-1) - 2a(n-r)$$

$$F''(r) = 2a + 2a > 0$$

F is concave-up $\Rightarrow$ so.

$$= \max\{F(1), F(n)\}$$

$$= \max\{a(n-1)^2 + an^3,\ a(n-1)^2 + an^3\}$$

$$= a(n-1)^2 + an^3$$

$$= an^3$$

$$\boxed{= O(n^3)}$$

2. (15 pts, standard) As we all know, mergesort is a procedure that can sort an array of n integers in time $O(n \log n)$ in best-, average-, worst- cases. We also know that partition is a procedure, with return value r, that reorganizes an array of n integers and splits the reorganized array into a low-part and a high-part, separated by the element indexed by r. Let mixsort$(A, 1, n)$ be an algorithm that sorts an array A with n integers. It works as follows:

```
mixsort(A, 1, n){
if n == 1, return;
r=partition(A, 1, n);
//run insertsort on the low part
insertsort(A, 1, r − 1);
//run mergesort sort on the high part
mergesort(A, r + 1, n);
}
```

Compute the average-case time complexity of mixsort. Please faithfully follow the procedure of "write a formula" and "directly compute the solution". Please also show your math work.

write a formula.

$$T_{Avg}(n) = \frac{1}{n}\sum_{r=1}^{n} \{ O((r-1)\log(r-1)) + O(n-r)^2 + O(n)]$$

$$= \frac{1}{n}\sum_{r=1}^{n} \{ a(r-1)\log(r-1) + a(n-r)^2 + an]$$

$$= \frac{a}{n} \{ \sum_{r=1}^{n} (r-1)\log(r-1) + \sum_{r=1}^{n} (n-r)^2 + n]$$

$$= \frac{a}{n} \{ \sum_{r=1}^{n} (r-1)\log(r-1) + \sum_{r=1}^{n} (r-1)^2 + n^2]$$

$$\leq \frac{a}{n} \cdot 2 \cdot \sum_{r=1}^{n} (r-1)^2 + an$$

$$\leq \frac{2a}{n} \cdot \frac{n^3}{3} + an$$

$$\leq O(n\log n)$$

3. (10 pts) We use 4com to denote an operation that sorts 4 numbers. Show that one needs at least $\log_{24} n!$ number of 4com operations to sort n distinct numbers. You must write down your reasoning clearly.

unsorted n #'s. $\log_{24} n!$ bits on the of ordeit

step I use 4 com, I can reduce at most $\log_{24} 4$ bits.

sorted n #s. 0 bits on the info of ordeing

at most $\log_{24} 4!$ bits unknow infomole on the order.

4com.

the order of 4 #'s.

$\frac{\log_{24} n!}{\log_{24} 4!}$ uses.

4. (10 pts, not hard) Let $A[1..n]$ be an array of n distinct numbers. Design a worst case $O(n \log n)$ time algorithm to decide whether there is an index i such that $A[n-i+1]$ is the i-th smallest in $A[1..n]$. You may simply use English to describe your algorithm. (Hint: the input to your algorithm is the array and the output is yes/no. Note that i is not part of the input and your algorithm is going to find the i.)

array A.

1. if $(i=1)$
2. then True
3. else [while $i<n$]
   1. do If $[A[i] \neq A[i+1]]$
   2. then $[i = i+1]$
   3. else False
4. output Yes/No.

they take $O(n \log n)$ times, the while loop is $n-1$ times
it is $O(1)$, if else is $O(1)$. Then the total use $O(n\log n)$

5. (10 pts, hard) Let A be an array of $2n$ distinct babies where there are n purple hair babies and there are n red hair babies. Design a linear time and in-place and one-pass algorithm to re-organize the babies in the array so that babies with purple hair are on odd positions and babies with red hair are on even positions (hence, the resulting array of babies will be: purple, red, purple, red, purple, red, ...). You may use either English or psuedo-code to describe your algorithm.

1, one pass: This means that each element should be handled only once. The algorithm can traverse all elements at most once. Once an element is accessed, the same element shouldn't be accessed again.

In place: Everything we do to the element should be done in the given array itself. You shouldn't use extra or temporary space to work with any elements.

Linear-time: Linear time means that the time complexity of the algorithm should be linear, which actually means that it should take $O(n)$ time for the n input.

Set the two pointers to be odd and num. odd starts with an index value of 0 and num of 1. when A[num] carries eachbaby, it is compared with Purple. If it's purple, put in array and odd don't change. If it's not, odd = odd + 2. put in to array.

6. (10 pts, easy. Homework problem where I only talk about the solutions in class.) Let A be an array of n distinct numbers. Describe a worst case linear time algorithm to find both the maximal element and the minimal element in the array using roughly $1.5 \cdot n$ comparisons.

```
n = mylist.size( );
min = mylist[0];
for (int i(1) ; i<n-1, i++]
{ if (min > mylist[i])
      { min = mylist [i];}
}
retun min ;
```

max is same as min. The idea of this algorithm is that if I have 100 numbers. what I want to do is divide those 100 numbers into two parts. I'm going to compare the two part once, and then I going - compare each part to find the max or min. So we have a total 50+49+49 = 148.

7. (10 pts, you need only five minutes) For each of the following statement, you need only mark yes or no; no reasoning needed.

Yes. A. If my algorithm's input is of three numbers n, p, q, then the size of input is at least n.

Yes B. I have a program that crashes on every input. Is the program an algorithm?

No. C. Last night, I invented a new algorithm that is two times faster than quicksort in worst-case. Is it true that my algorithm has lower worst case time complexity than quicksort has?

Yes D. When we design algorithms to sort n numbers, can we assume that those numbers are of 32-bits?

No. E. Space complexity of an algorithm is always at least linear since input itself takes memory. Is this correct?