EECS4020 Algorithms

Homework 1
(On a voluntary basis: No need to submit)

You are encouraged to work in groups. If you wish, you may submit your solution (of any question) to our tutors on or before March 17, 2022, so that they may have a chance to give feedback of your work.

I. Getting Started

1. Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.

   (a) List the five inversions of array $(2, 3, 8, 6, 1)$.

   (b) What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?

   (c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

   (d) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (*Hint:* Modify MergeSort.)

2. You have just finished sorting an array $A[1..n]$ of n distinct numbers into increasing order. When you go out to have a break, your mischievous friend, John, has divided your array into two parts $A_{\mathsf{left}} = A[1..i]$ and $A_{\mathsf{right}} = A[i+1..n]$, and re-arrange the array so that he puts A_{right} in front of A_{left}; precisely, the array now becomes $A[i+1..n]A[1..i]$. See Figure 1 for an example.

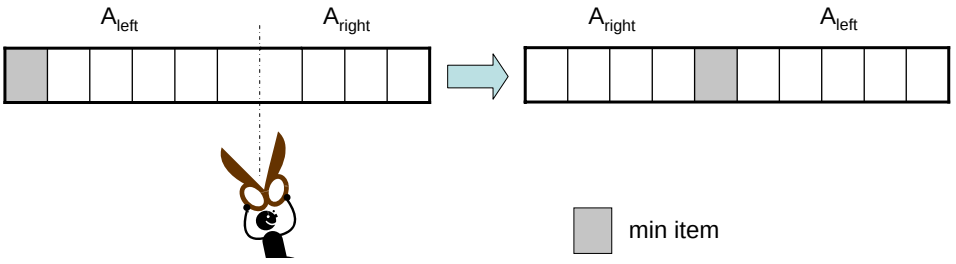

Figure 1: John's modification to the array.

After you come back, John tells you about what he has done, but without telling you the value of i. To reverse the change, you want to locate the entry with the minimum item, as this will be the boundary between A_{right} and A_{left}.

Design an $O(\log n)$-time algorithm to find the position of the minimum item. Show that your algorithm is correct.

3. Consider the following code `ComputeCount`:

```
ComputeCount()
1. Input a positive integer n;
2. Set count = 0;
2. for j = 1, 2, ..., n
3.     if j is a factor of n
4.     { Update count to become 1 - count; }
5. Output count;
```

The above code computes the value of `count` in $\Theta(n)$ time. Design a faster algorithm that can compute `count`, and analyze its running time. Explain why your algorithm is correct.

II. Growth of Functions

1. Given that

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0, \quad \text{where } a_m > 0.$$

Show that $f(n) = \Theta(n^m)$.

2. Show that $k \ln k = \Theta(n)$ implies $k = \Theta(n / \ln n)$.

3. What is wrong with the following argument?

"Since $n = O(n)$, and $2n = O(n)$, ..., we have

$$\sum_{k=1}^{n} k \cdot n = \sum_{k=1}^{n} O(n) = O(n^2)."$$

4. Someone wrote down the formula

$$O(f(n)) - O(f(n)) = 0.$$

What was his/her mistake? What should be the right hand side of the formula?

5. Arrange the following functions by order of growth, in ascending order. Also, partition the functions into equivalence classes such that the function $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. Here, we use lg to denote $\log_2$, and ln to denote $\log_e$.

e^n	$(\lg n)^{\lg n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$4^{\lg n}$	$(n+1)!$	$n \lg n$	n^3	$\lg^2 n$	$\lg(n!)$
$\sqrt{\lg n}$	$2^{\sqrt{2 \lg n}}$	2^{2^n}	$n^{1/\lg n}$	$\ln \ln n$	$n \cdot 2^n$
n	2^n	$n^{\lg \lg n}$	$\ln n$	1	$2^{\lg n}$

III. Solving Recurrences

1. Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n-1) + T(n/2) + n$. Use the substitution method to verify your answer.

2. Give a tight asymptotic bound for

$$T(n) = T(n/3) + T(2n/3) + n.$$

3. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2\,T\,(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

   is $T(n) = n \log_2 n$.

4. Give an asymptotic upper bound for $T(n)$ in each of the following recurrence. Make your bounds as tight as possible.

   (a) $T(n) = 9\ T(n/2) + n^3$
   (b) $T(n) = 7\ T(n/2) + n^3$
   (c) $T(n) = T(\sqrt{n}) + \log n$
   (d) $T(n) = 0.5\ T(n/2) + n$
   (e) $T(n) = 3\ T(n/3) + n/3$
   (f) $T(n) = 3T(n/2) + n \log n$

IV. Heapsort

1. Give an $O(n \log k)$-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists.

2. Given a list of n nearly sorted numbers, where each number is located within d positions from its correct location, show how to sort the numbers in $O(n \log d)$ time.