

# CS 3343 (Spring 2008) Assignment 5

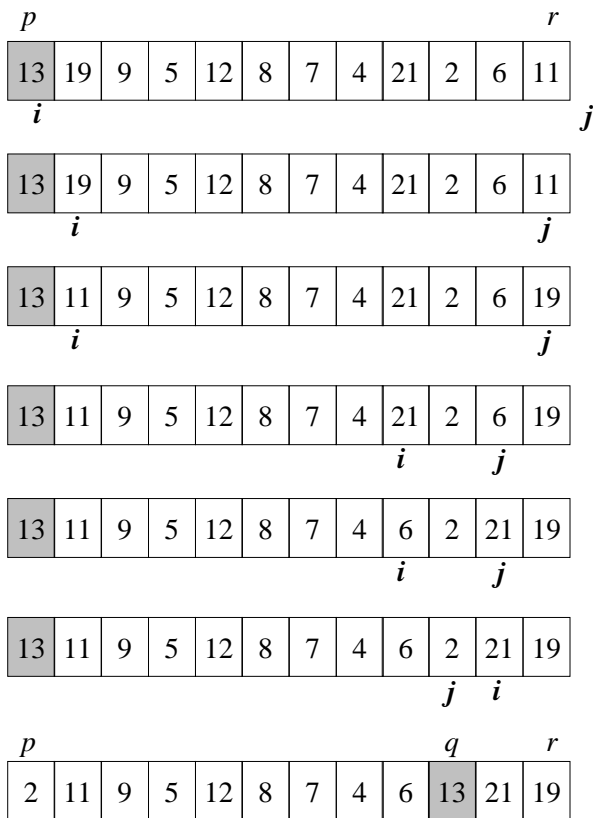
## Solution

1. (40 points) Quick sort.

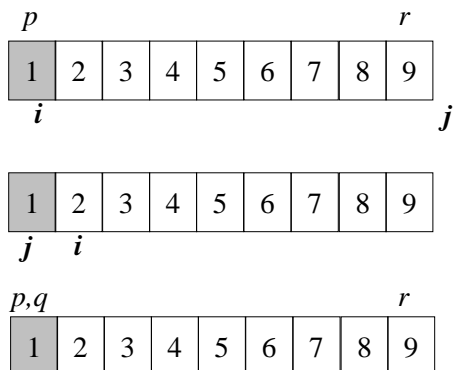
**Note:** the pseudocode given in the original lecture slides were buggy, since  $i, j$  may be out of bound. To fix the bug, we change the two *until* statements as below (also note the equal sign, which was missed in my correction email):

$until\ A[i] > x; \Rightarrow until\ i \geq r\ or\ A[i] > x;$   
 $until\ A[j] < x; \Rightarrow until\ j \leq p\ or\ A[j] < x;$

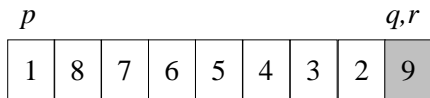
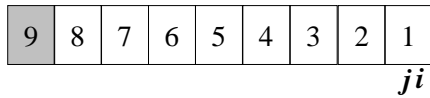
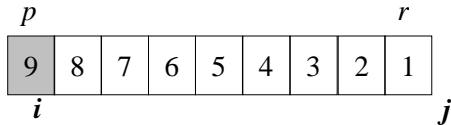
a. (10 points) Study the pseudocode of the Partition algorithm on Slide #12 (Lecture 11). Use Slide #13 as a model, illustrate the operation of Partition on array  $A = [13\ 19\ 9\ 5\ 12\ 8\ 7\ 4\ 21\ 2\ 6\ 11]$ . Importantly, indicate where is the pivot element when the algorithm terminates. There is no need to use colors.



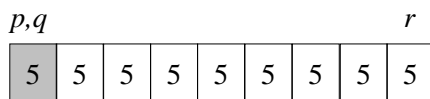
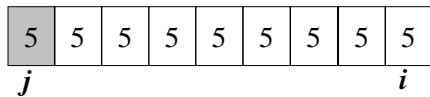
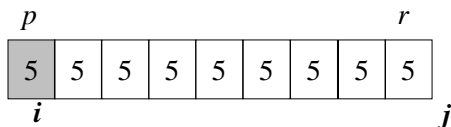
b. (5 points) Redo 1(a) on array  $B = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$ .



- c. (5 points) Redo 1(a) on array  $C = [9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1]$ .



- d. (5 points) The partition algorithm may result in extremely unbalanced partition when all elements are equal. Illustrate this by redoing 1(a) on array  $D = [5\ 5\ 5\ 5\ 5\ 5\ 5\ 5\ 5]$ . Can you find an easy fix so that the partition will be more balanced for this type of input? You can achieve this by modifying two lines in the pseudocode (without adding any lines).



To have the partition more balanced when dealing with an array with identical elements, We can change the two *until* statements to:

$until\ i \geq r\ or\ A[i] > x; \Rightarrow until\ A[i] \geq x;$

$until\ j \leq p\ or\ A[j] < x; \Rightarrow until\ A[j] \leq x;$

- e. (5 points) In the pseudocode for Quicksort on Slide #7, Quicksort was recursively applied to two subarrays: one subarray from index  $p$  to  $q - 1$ , and the other from index  $q + 1$  to  $r$ . This is correct because element  $A[q]$  is already in the correct position. Now consider the following modified pseudocode for Quicksort:

```

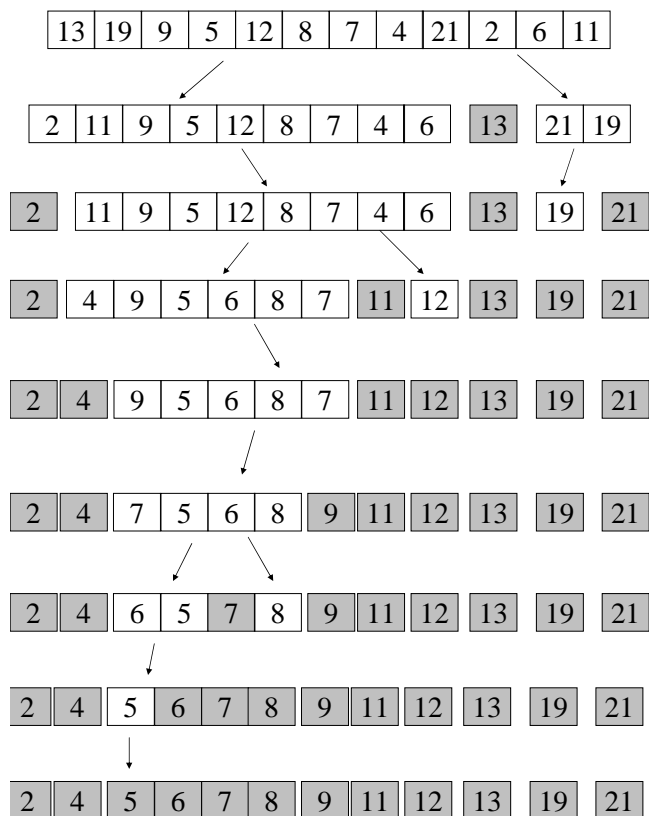
QUICKSORT(A, p, r)
  if (p < r)
    then q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q-1)
    QUICKSORT(A, q, r)

```

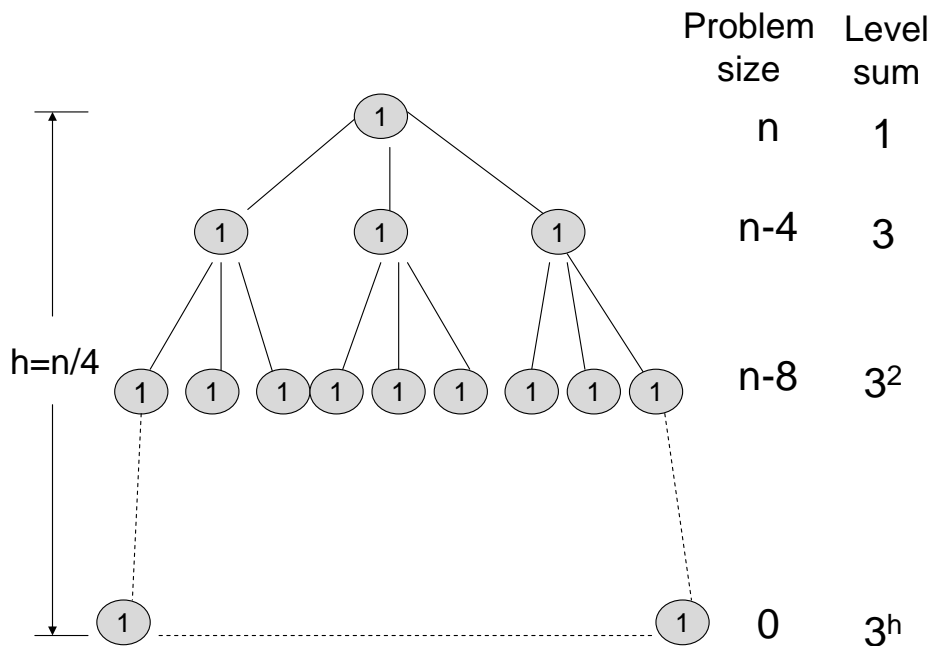
Assume that the function Partition is the same as the one on Slide #12. The only difference between the pseudocode above and the original one is the last line. In the above pseudocode, the second subarray starts from  $q$  instead of  $q + 1$ . Besides having to sort one extra element, this modified algorithm will sometimes fail to terminate. Give an example array for which the modified algorithm will fail to terminate.

**Answer:** Take the array in 1(b) as an example. After returning from Partition,  $p = q$ . Therefore, the first subarray  $A[p, q - 1]$  will be empty and the second subarray  $A[q, r]$  will have the same size as the original array  $A[p, r]$ . As a result, the recursive call will not terminate because the size of the subproblem is the same as that of the original problem.

- f. (10 points) Use Slide #42 as a model, illustrate the operation of Quick Sort (the version in lecture slides #7 and #12) on array A.



2. (20 points) Assuming  $T(0) = \Theta(1)$ , use **recursion tree method** to solve  $T(n) = 3T(n - 4) + 1$ .



$$T(n) = 1 + 3 + 3^2 + \dots + 3^h = \Theta(3^h) = \Theta(3^{n/4})$$

$$= \Theta((3^{1/4})^n) = \Theta(1.3^n)$$

3. (20 points) Assuming  $T(0) = \Theta(1)$ , use **recursion tree method** to solve  $T(n) = T(n/2) + T(n/3) + n$ . Prove that your solution is correct using the **substitution method** (you can just prove the Big-Oh part).

Hint: you will need the fact that  $(a+b)^k = \sum_{i=0}^k \binom{k}{i} a^{k-i} b^i$ , where  $\binom{k}{i} = \frac{k!}{i!(k-i)!}$ . For example,

$$\left(\frac{1}{2} + \frac{1}{3}\right)^3 = \frac{1}{2^3} + 3 \cdot \frac{1}{2^2} \cdot \frac{1}{3} + 3 \cdot \frac{1}{2} \cdot \frac{1}{3^2} + \frac{1}{3^3}.$$

The recursion tree is shown on the next page. The main difficulties in using the recursion tree method here are: (1) the left and right branches of the tree have different heights; and (2) the level sums are not obvious to compute. In particular, we have to find out whether the level sums are constant, decreasing, or increasing, and if increasing (decreasing), whether it is increasing (decreasing) geometrically or arithmetically.

By following the values on the tree nodes for a few levels, if you are careful enough, you can find that the level sums is a geometrically decreasing series (before the right branches ends). Take the second level as an example, the sum is  $\frac{n}{4} + 2 \cdot \frac{n}{6} + \frac{n}{9} = n((\frac{1}{2})^2 + 2 \cdot \frac{1}{2} \cdot \frac{1}{3} + (\frac{1}{3})^2) = n(\frac{1}{2} + \frac{1}{3})^2 = n(\frac{5}{6})^2$ . Similarly, the sum at the third level is  $\frac{n}{8} + 3 \cdot \frac{n}{12} + 3 \cdot \frac{n}{18} + \frac{n}{27} = n((\frac{1}{2})^3 + 3 \cdot (\frac{1}{2})^2 \cdot \frac{1}{3} + 3 \cdot (\frac{1}{3})^2 \cdot \frac{1}{2} + (\frac{1}{3})^3) = n(\frac{1}{2} + \frac{1}{3})^3 = n(\frac{5}{6})^3$ . Since the sum of a decreasing geometric series is dominated by the first term, we can guess that the total cost is dominated by the value in the root node, which is  $\Theta(n)$ .

After  $h_2$  levels, however, the level sum is not a geometric series any more, since each level has less and less number of nodes. Therefore, the recursion tree method only gives us a rough guess. We need to prove  $T(n) \in \Theta(n)$  using the substitution method. We will first prove  $T(n) \in O(n)$ . According to the definition of Big-oh, this means there must exist a positive  $c$  and  $n_0$  such that  $T(n) \leq cn$  for all  $n > n_0$ .

**Proof:**

Assume that  $T(k) \leq ck$  for all  $k < n$ . This means  $T(n/2) \leq cn/2$  and  $T(n/3) \leq cn/3$ . Substitute  $T(n/2)$  and  $T(n/3)$  into the recurrence, we obtain

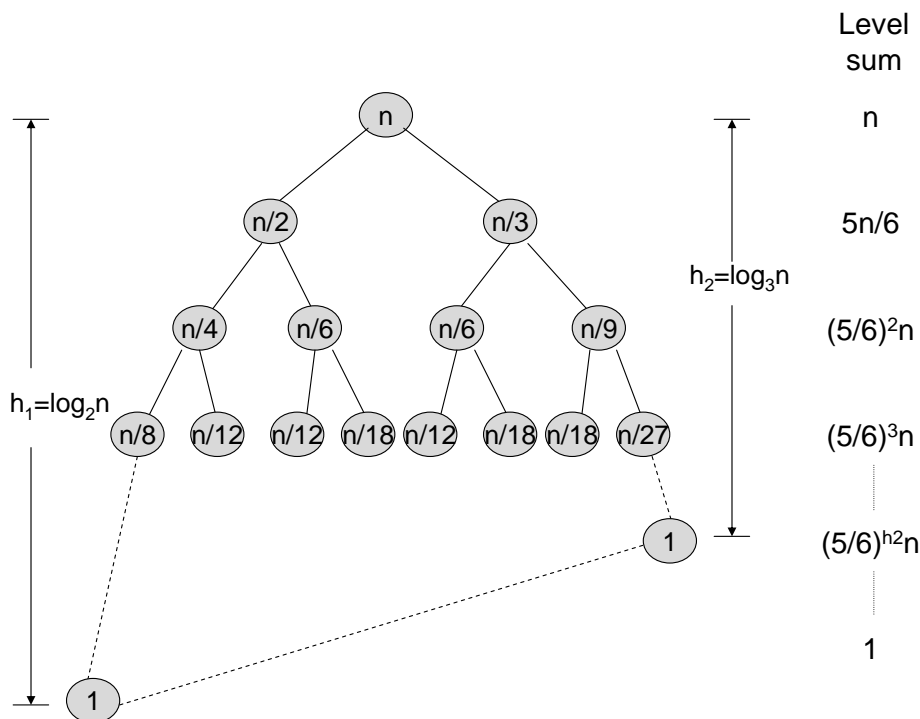
$$T(n) = T(n/2) + T(n/3) + n \leq cn/2 + cn/3 + n = 5cn/6 + n = cn + (n - cn/6).$$

If we choose  $c > 6$ , we have  $(n - cn/6) < 0$ , which implies  $T(n) \leq cn$ . Therefore,  $T(n) \in O(n)$ .

Similarly, we can prove  $T(n) \in \Omega(n)$ . According to the definition of  $\Omega$ , this means  $T(n) \geq dn$  for some positive  $d$  and all  $n > n_0$ . Assume that  $T(n/2) \geq dn/2$  and  $T(n/3) \geq dn/3$ . Substitute into the recurrence, we obtain

$$T(n) = T(n/2) + T(n/3) + n \geq dn/2 + dn/3 + n = 5dn/6 + n = dn + (n - dn/6).$$

If we choose  $d < 6$ , we have  $(n - dn/6) > 0$ , which implies  $T(n) \geq dn$ . Therefore,  $T(n) \in \Omega(n)$ . Since  $T(n) \in O(n)$  and  $T(n) \in \Omega(n)$ , we conclude that  $T(n) \in \Theta(n)$ .



4. **Bonus** (5 points) How much time did you spend on this homework? Who did you discuss with and what was the discussion about? What do you think about the difficulty level of the homework (harder than expected? just all right? easy?) What is the most difficult part? Do you have any comments/suggestions about the lecture, recitation, and homework?

**Note:** Nobody will get more than 80 points for this homework, which means if you have answered all the other questions correctly, answering these bonus questions will NOT help you earn any extra points.