CS3230
Tutorial 3

Q2. Given an array A of integers and a number k, give an efficient algorithm to decide if there exist two elements $A[i]$ and $A[j]$, $i \neq j$, such that $A[i] + A[j] = k$. Give time complexity bound of your algorithm.

Ans: (A) Sort the array (takes $O(n \log n)$ time).

(B) For each j, use binary search to check if the array contains the element $k - A[j]$ (as $A[i]$, with $i \neq j$).

Time complexity: $O(n \log n)$ for step (A) above. $O(\log n)$ for each j in step (B), and thus $O(n \log n)$ for step (B).

Alternatively, a better way to do step (B) is as follows:

Suppose array $A[1 : n]$ is sorted.

$i = 1$; $j = n$.

While $i < j$ do {

  If $A[i] + A[j] = k$, then output yes.

  Else If $A[i] + A[j] < k$, then $i = i + 1$.

  Else If $A[i] + A[j] > k$, then $j = j - 1$.

}

The above will do step (B) in $O(n)$ time. However, the overall complexity of steps (A) and (B) still remains $O(n \log n)$.

Q3. Consider the following modification of the partition algorithm done in class. Show that it works correctly.

Partition(A, i, j)

Assumption: $i < j$

1. Let $m = i + 1$, $n = j$;
2. While $m \leq n$, do {
   3. While $A[m] < A[i]$ and $m \leq n$ do { $m = m + 1$ }
   4. While $A[n] \geq A[i]$ and $n \geq m$ do { $n = n - 1$ }
   5. If $n > m$, then swap($A[m]$,$A[n]$).

   }
6. swap$(A[i], A[n])$
7. Return n.

End

Ans:

First note that m is monotonically non-decreasing and n is monotonically non-increasing throughout the algorithm.

Invariants of the algorithm during the while loop (steps 2–5)

> I1: $A[i+1], \ldots, A[m-1]$, are $< A[i]$
> I2: $A[n+1], \ldots, A[j]$, are $\geq A[i]$

Clearly, the invariants hold at the beginning of the first iteration of the while loop at step 2, as at that time $m = i+1$ and $n = j$.

Furthermore, whenever value of m or n is changed in steps 3 and 4, the invariants are maintained.

Also,

> While loop at step 3, increases m until $A[m] \geq A[i]$ or m becomes $> n$. Thus, at the end of the while loop at step 3, either $m > n$, or $A[m] \geq A[i]$.
>
> Similarly, while loop at step 4, decreases n until $A[n] < A[i]$, or $n < m$. Thus, at the end of the while loop at step 3, either $m > n$, or $A[n] < A[i]$.

Thus, at the end of any iteration of the while loop (of step 2), either $m > n$, or ($A[m] < A[i]$ and $A[n] \geq A[i]$) — in the later case, in the next iteration of the while loop of step 2, m will increase and n will decrease.

Thus, m increases in value after very iteration of the while loop in step 2 (except maybe for the first iteration), and n decreases in value after very iteration of the while loop in step 2 (except maybe for the first iteration).

It follows that eventually, $m > n$ and the while loop at step 2 will terminate. Using the invariants, we immediately have that at this point, $m = n+1$, $A[i+1], \ldots, A[n]$ are $< A[i]$ and $A[m], \ldots, A[j]$ are $\geq A[i]$.

Thus, step 6 correctly places the pivot value at $A[n]$, and at the end, $A[i], \ldots, A[n-1]$ are smaller than $A[n]$, and $A[n+1], \ldots, A[j]$ are $\geq A[n]$, as needed.

Q4. Given as input a sorted array A, containing n elements, and two numbers ℓ and u (where $\ell \leq u$). Give an algorithm to find how many numbers are there in the array which are between ℓ and u (both inclusive). That is, find the number of x in the array A such that $\ell \leq x \leq u$.

What is the time complexity of your algorithm.

Ans: (A) Suppose the array indices are from 1 to n.

(B) Use binary search to find least i such that $A[i] \geq \ell$ (if none, then we take $i = n+1$).

(C) Use binary search to find largest j such that $A[j] \leq u$ (if none, then we take j to be 0).

(D) Output $j - i + 1$.

Complexity of the algorithm: Steps (B) and (C) take $O(\log n)$ time. Rest of the operations take constant time. So the algorithm runs in time $O(\log n)$.

Q5. Suppose we are given an array of n integers between 1 to m (both inclusive). Preprocess the array such that one can answer the following query in constant time:

How many numbers are there in the array which are between ℓ and u (both inclusive), where $1 \leq \ell \leq u \leq m$.

What is the time complexity of your preprocessing algorithm? Try to make it linear in m and n.

Ans: (A) Use the idea of counting sort to obtain $C[1], C[2], \ldots$, where $C[i]$ gives number of elements in the array which are $\leq i$.

(B) Then, the output is $C[u] - C[\ell - 1]$, where we take $C[0] = 0$.

Complexity is: $O(m + n)$