# Problem Set: Searching and Sorting

**Problem 1.** Suppose you are given an array $A$ of $n$ distinct positive integers. Write an algorithm that will output one element $x$ of $A$ such that $x$ is among the top 75% elements of $A$. Note that the top 75% elements of $A$ are the last 75% elements when $A$ is sorted. Your algorithm should not take more than $0.25n$ comparisons. Only comparisons count! All other arithmetic/memory operations are free.

**Problem 2.** Given an array $A$ of $n \geq 10000$ distinct positive integers. Write an algorithm that will output one element $x$ of $A$ such that $x$ is not among the top 5 elements of $A$, neither is it among the bottom 5 elements of $A$. Note that the top and bottom 5 elements of $A$ are the first 5 and the last 5 elements when $A$ is sorted. Your algorithm should not take more than say 50 comparisons. Only comparisons count! All other arithmetic/memory operations are free.

**Problem 3.** Suppose we are given two $n$-element sorted arrays $A$ and $B$ that should not be viewed as sets (that is, $A$ and $B$ may contain duplicate entries). Describe an $O(n)$-time method for computing an array representing the set $A \cup B$ (with no duplicates).

**Problem 4.** Given an array of numbers, $(x_1, x_2, ..., x_n)$, the **mode** is the value that appears the most number of times in this array. Give an efficient algorithm to compute the mode for an array of $n$ numbers. What is the running time of your method?

**Problem 5.** Suppose you're given an integer array $A$ of $n$ integers. You want to figure out the number of duplicates (an integer that appears more than once) in $A$.

**Algorithm 1** : Counting Duplicates

$count \leftarrow 0$
**for** $i = 1$ to $n$ **do**
  **for** $j = i + 1$ to $n$ **do**
    **if** $A[i] = A[j]$ **then**
      $count \leftarrow count + 1$
**return** $count$

i. How many comparisons the above algorithm perform?

ii. Devise a better algorithm for this problem.

**Problem 6.** Suppose you are given an array $A$ of $n$ distinct integers and an integer $z \notin A$. You want to figure out the number of pairs in the array that sum to $z$. Again there's a simple $\frac{n(n-1)}{2}$ algorithm for solving this problem (which is very similar to the above duplicate counting algorithm). Devise a better algorithm for this problem.

**Problem 7.** We can rewrite insertion sort as a recursive algorithm. That will work as follows: Given an array $A$ of $n$ distinct integers, we recursively sort $A[1 \dots n-1]$ and then insert $A[n]$ into the left part (which is already sorted). Suppose to insert $A[n]$ you use linear search (we could use binary search though).
*Note:* This approach is sometimes called 'decrease and conquer' rather than 'divide and conquer', as we decrease or reduce problem instance to a smaller instance of the same problem and conquer by extending solution of smaller instance to obtain solution tp original problem.

i Write details of this algorithm in the pseudocode as given above (carefully state its base cases(s)).

ii Express your algorithm's runtime (number of comparisons) as a recurrence relation.

iii Using the recursion tree approach find a closed form for the runtime of your algorithm.

**Problem 8.** Compare the time efficiency of the below given iterative merge sort algorithm with the recursive approach discussed in class.

Following is the pseudocode for iterative merge sort where $A$ is the array to be sorted and *length* is the length of the array:

**Algorithm 2** : Iterative Implementation of MergeSort

```
if length < 2 then
    return                                  ▷ the array is already sorted
step ← 1
while step < length do
    stL ← 0
    stR ← step
    while stR + step ≤ length do
        Merge(A, stL, stL + step, stR, stR + step)
        stL ← stR + step
        stR ← stL + step
    if stR < length then
        Merge(A, stL, stL + step, stR, length)
    step ← step × 2
```

And the following is the pseudocode for the merge operation used in the previous algorithm:

**Algorithm 2** Merge

```
function MERGE(A, lSt, lEnd, rSt, rEnd)
    Left ← A[lSt, ..., lEnd]        ▷ Copies elements of A from lSt to lEnd to Left
    Right ← A[rSt, ..., rEnd]
    i ← 0
    j ← 0
    for k = lSt to rEnd do
        if Left[i] ≤ Right[j] then
            array[k] ← Left[i]
            i ← i + 1
        else
            array[k] ← Right[j]
            j ← j + 1
        if i = lEnd + 1 − lSt then
            for l = k + 1 to rEnd do
                array[l] ← Right[j]
                j ← j + 1
            return
        if j = rEnd + 1 − lSt then
            for l = k + 1 to rEnd do
                array[l] ← Left[i]
                i ← i + 1
            return
```

**Problem 9.** Given a list of $n$ distinct positive integers, partition the list into two sub-lists, each of size $\frac{n}{2}$, such that the difference between the sums of the integers in the two sub-lists is maximized. Write a $O(n \log n)$ recursive algorithm for this problem. You may assume that $n$ is a multiple of 2.

**Problem 10.** Suppose we are given the GPA of a set of students and we want to find the top $k - th$ percentile of students. Devise an algorithm to solve this problem in the smallest possible time complexity (number of comparisons).

**Problem 11.** Show that in order to find the maximum and minimum of an array, in the worst case $\frac{3n}{2} - 2$ comparisons are required.
*Hint:* Check how many numbers are there that are either minimum or maximum and how is their count effected by a comparison. It should be noted that there would be one maximum and minimum at the end of the algorithm.

**Problem 12.** Suppose you are given an array $A[1...n]$ of sorted integers that has been circularly shifted $k$ positions to the right. For example, $[35, 42, 5, 15, 27, 29]$ is a sorted array that has been shifted $k = 2$ positions. We can obviously find the largest element in $O(n)$ time. Describe an $O(\log n)$ time algorithm to find the maximum in $A$.

**Problem 13.**
1. Given two sets $X$ and $Y$, devise an efficient algorithm to determine whether $X$ and $Y$ are disjoint, i.e. their intersection is zero. Analyze the complexity of your algorithm in terms of $|X|$ and $|Y|$. Don't forget to consider all the cases of sizes of arrays.

2. Devise an efficient algorithm to compute the union of sets $X$ and $Y$ where $m = max(|X|, |Y|)$. The output should be a single array of distinct elements that forms the union of the two sets.

   (a) Assume that X and Y are unsorted. Give an $O(m \log m)$ algorithm for the problem.

   (b) Assume that X and Y are sorted. Give an $O(m)$ algorithm for the problem.

**Problem 14.** Given a binary string (string of 1's and 0's), count the number of substrings that start and end with a 1.

**Problem 15.** Suppose the number of inversions in an array $A$ of size $n$ is 10, where $10 < n$. Which of the two is more suitable for sorting $A$: Insertion Sort or Bubble Sort. Explain your answer.