

Homework 1

1. For each of the following problems, give an algorithm that finds the desired numbers within the given amount of time.

- (a) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *maximizes* $|x - y|$. Your algorithm must run in $O(n)$ worst-case time.

```

let  $min = 0, max = \infty$ 
for  $value \in S$ 
    if  $value < min$  then  $min = value$ 
    if  $value > max$  then  $max = value$ 
return  $(max, min)$ 

```

- (b) Let S be a *sorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *maximizes* $|x - y|$. Your algorithm must run in $O(1)$ worst-case time.

```

let  $max = S[n - 1]$ 
let  $min = S[0]$ 
return  $(max, min)$ 

```

- (c) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *minimizes* $|x - y|$, for $x \neq y$. Your algorithm must run in $O(n \log n)$ worst-case time.

Sort S using heapsort
Follow the algorithm used in part d.

- (d) Let S be a *sorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *minimizes* $|x - y|$, for $x \neq y$. Your algorithm must run in $O(n)$ worst-case time.

```

let  $index1 = 0, index2 = 0, minDifference = \infty$ 
for  $i \in [0, n - 1]$ 
    if  $(S[i + 1] - S[i]) < minDifference$ :
        then  $index1 = i + 1, index2 = i$ 
return  $(S[index1], S[index2])$ 

```

- Given two sets S_1 and S_2 (each of size n), and a number x , describe an $O(n \log n)$ algorithm for finding whether there exists a pair of elements, one from S_1 and one from S_2 , that add up to x . (For partial credit, give a $\Theta(n^2)$ algorithm for this problem.)

Sort S_1 using heapsort	$O(n \log n)$
for $n \in S_2$	$O(n)$
let $\text{diff} = x - n$	$O(1)$
Perform a binary search on S_1 for diff	$O(\log n)$
if it is found then return (diff, n)	$O(1)$

This is $O(n \log n) + O(n \log n) = O(n \log n)$

- Devise an algorithm for finding the k smallest elements of an *unsorted* set of n integers in $O(n + k \log n)$.

Using a max-heap limited to size k , we can loop through each item $O(n)$ in the set and compare it to the maximum element in the heap $O(1)$. If the element is less than the max, then replace the maximum in the heap $O(1)$. At the end, the elements in the max-heap will be the k smallest elements. This is $O(n)$.

- Mr. B. C. Dull claims to have developed a new data structure for priority queues that supports the operations Insert, Maximum, and Extract-Max — all in $O(1)$ worst-case time. Prove that he is mistaken.
- Suppose that you are given a sorted sequence of distinct integers a_1, a_2, \dots, a_n . Give an $O(\log n)$ algorithm to determine whether there exists an i index such as $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$. In $\{2, 3, 4, 5, 6, 7\}$, there is no such i .