**Algorithm: Lab5 (By Sujiv Shrestha ID:610145)**

**Problem 1.**

1. Show all steps of QuickSort in sorting the array [1, 6, 2, 4, 3, 5]. Use leftmost values as pivots at each step.

**Problem2**

2. Show all steps of In-Place QuickSort in sorting the array [1, 6, 2, 4, 3, 5] when doing first partition. Use leftmost values as pivots.

Step 1: k=0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 6 | 2 | 4 | 3 | 5 |
| 0 | 1 | 2 | 3 | 4 | 5 |

↑ **Pivot**

Step 2: Swap kth element with rightmost element (rth)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | 6 | 2 | 4 | 3 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 |

↑ **Pivot**

Step 3: x = 1 (pivot element)

Step 4: in-place partition and get position of pivot point i = 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 5 | 6 | 2 | 4 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 |

↑ **Pivot**

**Problem3**

3. In our average case analysis of QuickSort, we defined a *good self-call* to be one in which the pivot *x* is chosen so that number of elements < x is less than 3n/4, and also the number of elements > x is less than 3n/4. We call an x with these properties a *good pivot.* When n is a power of 2, it is not hard to see that at least half of the elements in an n-element array could be used as a good pivot (exactly half if there are no duplicates). For this exercise, you will verify this property for the array A = [5, 1, 4, 3, 6, 2, 7, 1, 3] (here, n = 9). Note: For this analysis, use the version of QuickSort in which partitioning produces 3 subsequences *L, E, R* of the input sequence *S.*

a. Which x in A are good pivots? In other words, which values x in A satisfy:

i. the number of elements < x is less than 3n/4, and also

ii. the number of elements > x is less than 3n/4

Answer:

Input Array:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 1 | 4 | 3 | 6 | 2 | 7 | 1 | 3 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Input Array in sorted order:

s

Good pivot points

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 3 | 3 | 4 | 5 | 6 | 7 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

The good pivots that satisfy i. and ii. are [3,3,4]

b. Is it true that at least half the elements of A are good pivots?

For this particular case it is not true that at least half the elements of A are good pivots.

**Problem4**

4. *Interview Question.* Give an o(n) (“little-oh”) algorithm for determining whether a sorted array A of distinct integers contains an element m for which A[m] = m. You must also provide a proof that your algorithm runs in o(n) time.

Solution:

**Algorithm** check(S, n)

**Input** sorted sequence S with n integers

**Output** true if A[m]=m otherwise false

mid←n/2

**if**(n≤1 and A[0] ≠ 0) **then**

**return** **false**

**if**(S[mid]=mid) **then**

**return** **true**

**else** **if**(S[mid]<mid) **then**

S1←S.copyRange(0, mid-1)

**return** check(S1,mid)

**else**

S2←S.copyRange(mid, n-1)

**return** check(S1,n-mid)

Proof: In the worst case when m = 0 where A[m] = m, the number of recursive calls are equal to the number of terms in sequence S: n/2, n/4, n/8, ……………, n/2m (= 1) [where m = logn ]. Hence, the running time for this algorithm in worst case is Θ(m) or Θ(logn). And we know that logn is o(n).

**Problem5**

5. Review of SubsetSum Problem: Given a set S = {s0, s1,s2, …, sn-1} of positive integers and a non-negative integer k, find a subset T of S so that the sum of the integers in T equals k or indicate no such subset can be found.

We have already seen a brute force solution to this problem in an earlier lab. In this exercise, you are going to come up with a recursive solution for SubsetSum. Write the pseudo code for your algorithm.