### Lab 3-1 A: Divide and Conquer Basics *Goal: Review Quick Sort and Quick Select*

1. Partition the following set of numbers using Lomuto’s algorithm using 20 as the partitioning element. Show the state of the array after each swap. You may need more or fewer rows

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 29 | 13 | 23 | 27 | 17 | 25 | 12 | 26 | 15 |
| 20 | 13 | 29 | 23 | 27 | 17 | 25 | 12 | 26 | 15 |
| 20 | 13 | 17 | 23 | 27 | 29 | 25 | 12 | 26 | 15 |
| 20 | 13 | 17 | 12 | 27 | 29 | 25 | 23 | 26 | 15 |
| 20 | 13 | 17 | 12 | 15 | 29 | 25 | 23 | 26 | 27 |
| 15 | 13 | 17 | 12 | 20 | 29 | 25 | 23 | 26 | 27 |

1. Give a **recurrence relation** that describes number of key comparisons that would occur in QuickSelect if the partitioning step splits the array into equal halves each time it is applied. Include the work done by the partitioning algorithm.

QS(n) = 2QS(n/2) + O(n - 1)

1. Give the **recurrence relation** that describes the worst case number of key comparisons that would occur in QuickSort. Include the work done by the partitioning algorithm.

QS(n) = QS(1) + QS(n - 1) + O(n - 1)

**When asked for an algorithm, provide pseudo code.**

1. Let S be an unsorted array of n integers. Give an algorithm that finds the pair of numbers x and y in S that maximizes |x-y|. Your algorithm must run in O(n) worst case time.

X <- QuickSelect(S[], 0)

Y <- QuickSelect(S[], size of S - 1)

Return (S[x], S[y])

1. Let S be a sorted array of n integers. Give an algorithm that finds the pair of numbers x and y in S that minimizes |x-y| for x ≠ y. Your algorithm must run in O(n) worst case time.

mindif = S[1] – S[0]

x = S[0]

y = S[1]

for i = 1 to n - 1 do

if S[i + 1] - S[i] < mindiff then

mindiff = S[i + 1] - S[i]

x = S[i]

y = S[i + 1]

return (x, y)

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

Sort S with quicksort //O(nlogn)

mindif = S[1] – S[0]

x = S[0]

y = S[1]

for i = 1 to n - 1 do

if S[i + 1] - S[i] < mindiff then

mindiff = S[i + 1] - S[i]

x = S[i]

y = S[i + 1]

return (x, y)

1. Given two sets of numbers S1 and S2 and a number x, , give an O(n \* log n) algorithm for finding whether there exists a pair of elements, one from S1 and one from S2, that add up to x.

PairSumToX(S1[left..right], S2[left..right], x)

Sort both sets with quicksort //O(nlogn)

### Lab 3-1 B: Divide and Conquer Problems *Goal: Practice in applying divide and conquer to problem solving*

1. **Entry = Index:** Suppose that you are given a sorted list of distinct integers {a1; a2; : : : an}. Give a divide-and conquer algorithm that determines whether there exists an index i such that ai = i. For example, in { -10;-4; 3; 41}, a3 = 3, but in {4; 7; 19; 20} there is no such i. State the recurrence relation for the running time and the running time.

**Algorithm:**

EntryIdx(S[], left, right)

If right >= 1 then

Mid = 1 + (right - 1) / 2

If S[mid] == mid then

Return mid

If S[mid] > mid

Return EntryIdx(S[], left, mid - 1)

Return EntryIdx(S[], mid + 1, right)

Return -1

**Analysis:**

Recurrence Relation: T(n) = T(n / 2) + O(1)

Asymptotic Running Time: O(log n)

1. Let M be an n x n matrix of integers in which the integers in every row are in increasing order from smallest row index to largest row index and the integers in every column is in increasing order from smallest column index to largest column index.   
   Design and efficient algorithm (e.g. O(n) ) that finds a given integer or concludes that the integer is not in the array.

## Example

int[][] board1 = new int[][] {

{1, 2, 8, 9},

{3, 6, 12, 13},

{7,10,13, 29},

{10, 11, 28, 30}

};

int target = 12;

**Algorithm**:

SearchMatrix(S[][], i, j, x) //first call i = 0 and j = n -1

If i < n and j >= 0

If S[i][j] == x

Return true

If S[i][j] > x

Return SearchMatrix(S[][], i, j-1, x)

Return SearchMatrix(S[][], i + 1, j, x)

Return false

**Analysis**:

Recurrence Relation: T(n) = T(n\*n / n) + O(1)

Asymptotic Running Time: O(n)

1. **Stock Price Analysis**: You are working for a small stock investment company that wants to look for patterns in optimal trading days in a given time period of n days. They want to find the best **pair** of days in a period of n days to buy a stock on the first day of the pair and sell it on the second day of the pair. That is, they want the biggest positive difference between the selling price on the second day and the buying price on the first day. Assume for simplicity that the buying and selling price on a given day are the same. Assume you know the stock price for every day.  
   Specify an Θ (n log n) **Divide and Conquer algorithm.** (Note: there are Θ (n) solutions but that is not what is asked for here)

Algorithm:

StockAnalysis(A[], low, high)

If low == high

Return (start, start)

Mid = (low + high) / 2

Left = StockAnalysis(A[], low, mid)

Right= StockAnalysis(A[], mid + 1, high)

min = low

For i from low to mid

If A[i] < A[min] then

min = i

max = mid

For i from mid to high do

If A[i] > A[max] then

Max = i

Return max(left, right, (min, max)) //return max diff of pairs

Analysis:

Recurrence Relation: T(n) = 2T(n / 2) + O(n)

Asymptotic Running Time: n log n