16

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Analysis and Design of Algorithms

CS2102
Divide and Conquer Practice
2020-II

Submission deadline: 15 Oct, 20:05
Number of questions: 5

- Write your C++ code inside the *answers* folder in order to generate a single PDF file. The *problem*{*1,2,3,4,5*}*.cpp* files should include the functions required by the problem and the main function. No need to create header or other source files per problem.
- Read the questions carefully and write your answers clearly. Answers that are not legible will not have any score.
- Notes are allowed. To compile this file you can use the command latexmk -pdf -shell-escape main.tex
- Consider edge cases properly, any file that doesn't compile or doesn't met the requirements will not have any grade (clang++ or g++ are preferred).
- **STD compiler flag**: -std=c++2a (replace this by your actual configuration)

Outcomes:

a. Apply appropriate mathematical and related knowledge to computer science.
b. Analyze problems and identify the appropriate computational requirements for its solution.

2-

Problem 1 (Outcomes a) - 5 points

Write a divide-and-conquer $\Theta(n * log(n))$ program to measure how far a sequence of n numbers is from being sorted in descending order.

Intuition: If we measure the number of changes we need to do in a list of numbers to be sorted in ascending order we need to count the number of pairs i, j such that $A[i] > A[j]$ and $i < j$. That count will give us the measure of how far a sequence of numbers is from being sorted. For example, using the previous definition the similarity measure for the array $\{2,4,1,3,5\}$ is 3 since (2,1), (4,1) and (4,3) are out of order.

```
#include <iostream>
#include <vector>

using namespace std;

///BASED ON MERGE SORT IMPLEMENTATION
```

```
void merge(vector<int> & numbers, vector<int> &temp, int min, int mid, int
 ↪  max,long & count){
    int i = min, j = mid, k = min;
    while ((i <= mid - 1) and (max >=j)) {

        if (numbers[i] <= numbers[j])
            temp[k++] = numbers[i++];

        else{
            temp[k++] = numbers[j++];
            count = count + (mid - i);
        }
    }

    while (i <= mid - 1)
        temp[k++] = numbers[i++];

    while (j <= max)
        temp[k++] = numbers[j++];

    for (i = min; i <= max; i++)
        numbers[i] = temp[i];
}


void mergeSort(vector<int> & numbers,vector<int> & temp, int min, int max,
 ↪  long &count) {
    int mid = (min + max) / 2;
    if (max > min) {
        mergeSort(numbers, temp, min, mid,count);
        mergeSort(numbers, temp, mid + 1, max,count);
        merge(numbers, temp, min, mid + 1, max,count);
    }
}

void countInvertions(vector<int> & numbers, long & count){
    vector<int> temp(numbers.size());
    mergeSort(numbers, temp, 0, (int)numbers.size() - 1, count);
}

long problem1(std::vector<int> & numbers) {
    long tobeReturned=0;
    countInvertions(numbers,tobeReturned);
    return tobeReturned;
}
```

counting inversion

```
int main() {
    std::cout << "Problem 1 - Similarity\n";
    vector<int> numbers = {2,4,1,3,5};
    std::cout << problem1(numbers) << "\n";
}
```

Problem 2 (Outcomes a) - 3 points

Write a divide-and-conquer $\Theta(n * log(n))$ program that counts the total number of times that a number k appears in a sequence of numbers.

```
#include <iostream>
#include <vector>



int problem2(std::vector<int> & numbers,int min,int max, int k){
    if(min>=max){
        if(numbers[min]==k)
            return 1;
        else
            return 0;
    }

    int dividedProblem1=problem2(numbers,min,(min+max)/2,k);
    int dividedProblem2=problem2(numbers,(min+max)/2+1,max,k);

    return dividedProblem1+dividedProblem2;
}

int main(){
    std::cout << "Problem 2 - Occurrences\n";

    std::vector<int> numbers = {5,5,5,5,5,6,10,5,2,5};
    int k = 5;
    std::cout << problem2(numbers,0,(int) numbers.size()-1, k) <<
     ↪  std::endl;
}
```

Problem 3 (Outcomes a) - 5 points

Consider the Volatile Chemical Corporation investing problem described in Cormen's

Chap 04. Write a divide-and-conquer $\Theta(n * log(n))$ program that receives an input array of n numbers representing the stock prices of each day and returns the days that maximizes our profit when buying and selling stocks.

```
#include <iostream>
#include <vector>

using namespace std;

pair<pair<int,int>,long>findMaxCross(vector<int> &stockPrices,int min,int
 ↪  mid ,int max){
    long sum=0;
    int posLeft=mid,posRight=mid;
    long rightSum=INT64_MIN;
    long leftSum=INT64_MIN;
    for(auto i=mid; i>=min;--i){
        sum+= stockPrices[i];
        if(sum>leftSum){
            leftSum=sum;
            posLeft=i;
        }
    }
    sum=0;


    for(auto i=mid; i<=max;++i){
        sum+= stockPrices[i];
        if(sum>rightSum){
            rightSum=sum;
            posRight=i;
        }
    }
    return make_pair(make_pair(posLeft,posRight),leftSum+rightSum);
}


pair<pair<int,int>,long> findMaxSubArray(vector<int> &stockPrices,int min
 ↪  ,int max){
    if(min==max)
        return make_pair(make_pair(min,max),stockPrices[min]);
    else{
        int mid=(min+max)/2;
        auto left = findMaxSubArray(stockPrices,min,mid);
        auto right = findMaxSubArray(stockPrices,mid+1,max);
        auto cross = findMaxCross(stockPrices,min,mid,max);
```

```
        if(left.second>=right.second and left.second>=cross.second) return
        ↪ left;
        else if(right.second>=left.second and right.second>=cross.second)
        ↪ return right;
        else return cross;
    }
}
```

```
pair<int,int> problem3(std::vector<int> & stockPrices){
    vector<int> changeFactor;
    changeFactor.push_back(0);
    for(auto i=1;i<stockPrices.size();++i){
        changeFactor.push_back(stockPrices[i]-stockPrices[i-1]);
    }
    return
    ↪ findMaxSubArray(changeFactor,0,(int)changeFactor.size()-1).first;
}

int main() {
    std::cout << "Problem 3 - Volatile investment\n";

    std::vector<int> stocks =
    ↪ {100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97};
    auto [buy, sell] = problem3(stocks);

    printf("Buying at day %d\nSelling at day %d\n", buy, sell);
}
```

Problem 4 (Outcomes a) - 3 points

Write a lineal (1pt) and a divide-and-conquer (2pt) $\Theta(log(n))$ program that compute x^n considering x as the base and n as the exponent.

```
#include <iostream>

long problem4BruteForce(long base, long power){
    long result=1;
    for(int i=0;i<power;++i)
        result*=base;
    return result;
}


long problem4DivideAndConquer(long base, long power){
```

```
    if(power==0)
        return 1;
    else{
        long extraPowerIfNeeeded = power%2==0 ? 1:base;
        long dividedProblem1 = problem4DivideAndConquer(base, power / 2);
        long dividedProblem2 = problem4DivideAndConquer(base, power / 2);
        return dividedProblem1 * dividedProblem2 * extraPowerIfNeeeded;
    }
}

int main() {
    std::cout << "Problem 4 - Powering\n";

    long base = 2;
    long power = 5;
    printf("%ld power of %ld is %ld\n", base, power,
     ↪  problem4BruteForce(base, power));
    printf("%ld power of %ld is %ld\n", base, power,
     ↪  problem4DivideAndConquer(base, power));
}
```

Problem 5 (Outcomes a, b) - 4 points

Write a recursive insertion sort algorithm (2pt) that receives an unsorted array of numbers, then express that algorithm using a recurrence relation and approximate T(n) (2pt).

```
#include <iostream>
#include <vector>
using namespace std;

void insertionSort(vector<int> &numbers, long index){
    if(index==1) return;
    else{
        index-=1;
        insertionSort(numbers,index);
        int value = numbers[index];
        long pos= index -1;

        while(pos>=0 && numbers[pos]>value){
            numbers[pos+1]=numbers[pos];
            pos-=1;
        }

        numbers[pos+1] = value;
    }
}
```

```
int main() {
    vector<int> stocks =
     ↪  {100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97};
    insertionSort(stocks,(long) stocks.size());

    for(auto i:stocks)
        cout<<i<<" ";

    std::cout << "\nProblem 5 - Recursive Insertion Sort\n";
    std::cout << "\nIt works :D\n";

    /// T(n) = T(n-1) + O(n), n>1
    /// T(n) = 1 , n<=1
```

}

$T(n) = 2T(n/2) + n$ (replace these lines with your answer)
$T(n) = O(n)$