In-class Lab Exercise

Week 5

Question 1

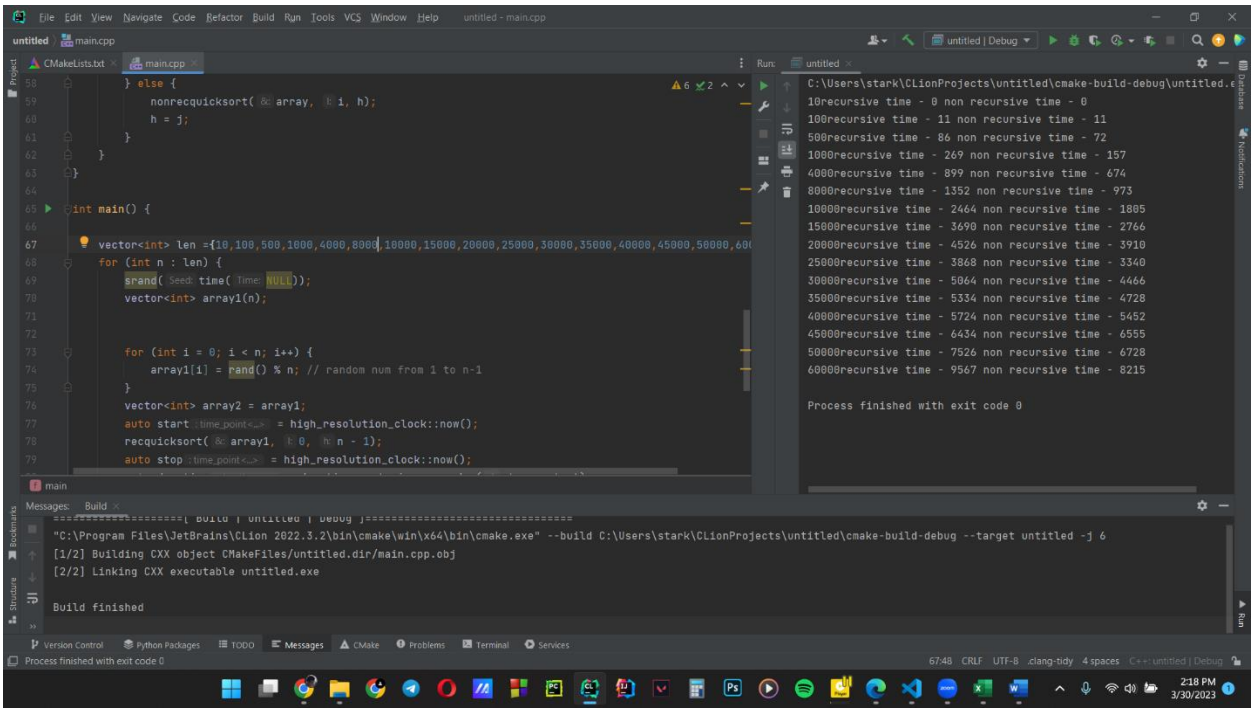
Recursive Quick sort Algorithm

```
int split(vector<int>& array, int l, int h) {
    int key = array[h];
    int i = l - 1;

    for(int j=l; j < h; j++) {
        if(array[j] < key) {
            i++;
            swap(array[i], array[j]);
        }
    }

    swap(array[i + 1], array[h]);
    return i+1;
}

void quicksort(vector<int>& array, int l, int h) {
    if(l < h) {
        int pi = split(array, l, h);

        quicksort(array, l, pi - 1);
        quicksort(array, pi + 1, h);
    }
}
```

Non-recursive quick sort Algorithm

```
void quicksort(vector<int>& array, int l, int h) {
    while (l < h) {
        int i = l, j = h;
        int key = array[(l + h) / 2];

        while (i <= j) {
            while (array[i] < key) {
                i++;
            }
            while (array[j] > key) {
                j--;
            }

            if (i <= j) {
                swap(array[i], array[j]);
                i++;
                j--;
            }
        }

        if (j - l < h - i) {
            quicksort(array, l, j);
            l = i;
        } else {
            quicksort(array, i, h);
            h = j;
        }
    }
}
```

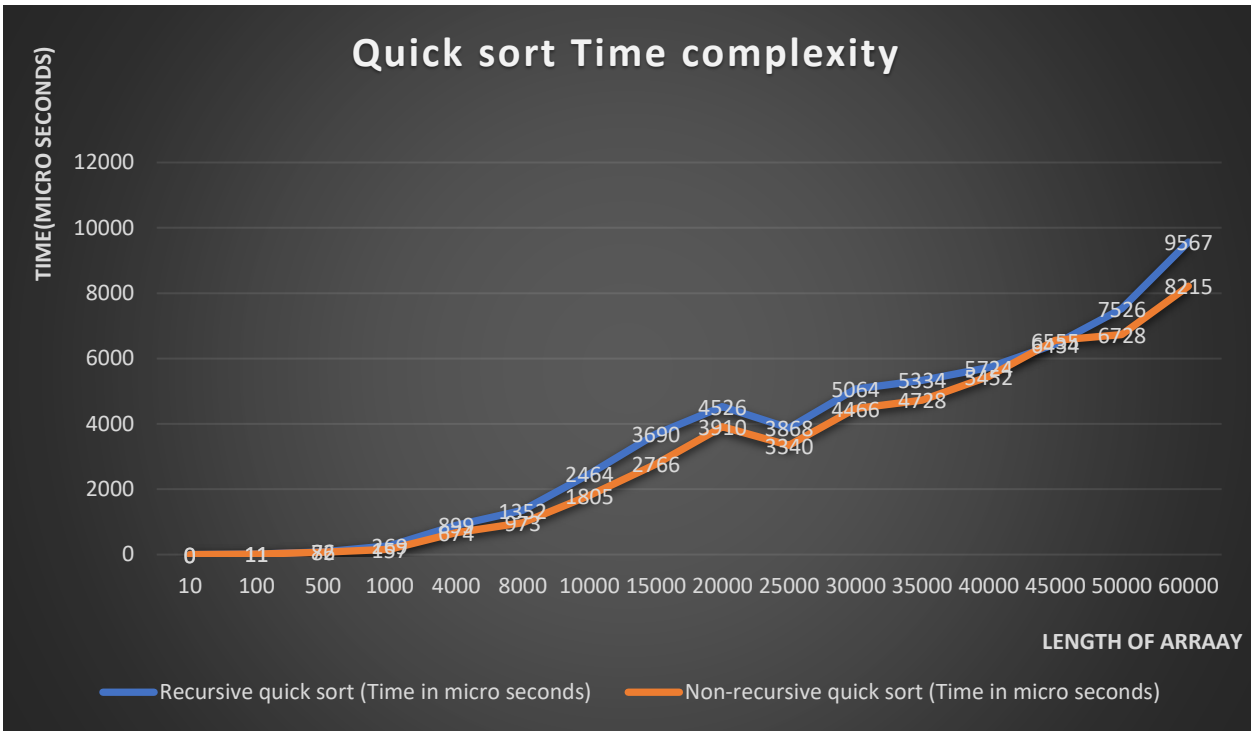
Terminal Output:-



Length of Array	Recursive quick sort (Time in micro seconds)	Non-recursive quick sort (Time in micro seconds)
10	0	0
100	11	11
500	86	72
1000	269	157
4000	899	674
8000	1352	973
10000	2464	1805
15000	3690	2766
20000	4526	3910

25000	3868	3340
30000	5064	4466
35000	5334	4728
40000	5724	5452
45000	6434	6555
50000	7526	6728
60000	9567	8215

Graph according to the data in the table



Discussion –

- For small sizes of arrays , recursive and non recursive implementations show same time results.
- But when we increase the number of elements in the arrays , recursive quick sort algorithm shows a much higher time taken .
- In theory the time complexity of both functions are O(nlog n).
- But when the input sizes get larger , the memory overhead becomes larger in recursive function

because , it stores every recursive step in a stack that means for larger arrays the memory usage of recursive function is much higher.

- Our gathered data confirms that .
- But recursive method is easier to implement in the programmer's side.

Question 2

Functions used for implementation

- Used quicksort algorithm

```
int split(vector<int>& array, int l, int h) {
    int key = array[h];
    int i = l - 1;

    for(int j=l; j < h; j++) {
        if(array[j] < key) {
            i++;
            swap(array[i], array[j]);
        }
    }

    swap(array[i + 1], array[h]);
    return i+1;
}

void recquicksort(vector<int>& array, int l, int h) {
```

```
    if(l < h) {
        int pi = split(array, l, h);

        recquicksort(array, l, pi - 1);
        recquicksort(array, pi + 1, h);
    }
}
```

- Created a function median(array)

```
float median(vector<int> array){
    int x = array.size();
    int y = array.size()/2;
    if (x==1)   //if size is 0 return only number
        return array[0];
    else if (x % 2 ==0) // if even elements should get 2
        // mid numbers in sorted array and get average of both
        return ((array[y-1]+array[y])/2.0);
    else // if odd elements get the middle number of sorted arr
        return array[y];
}
```

- Main

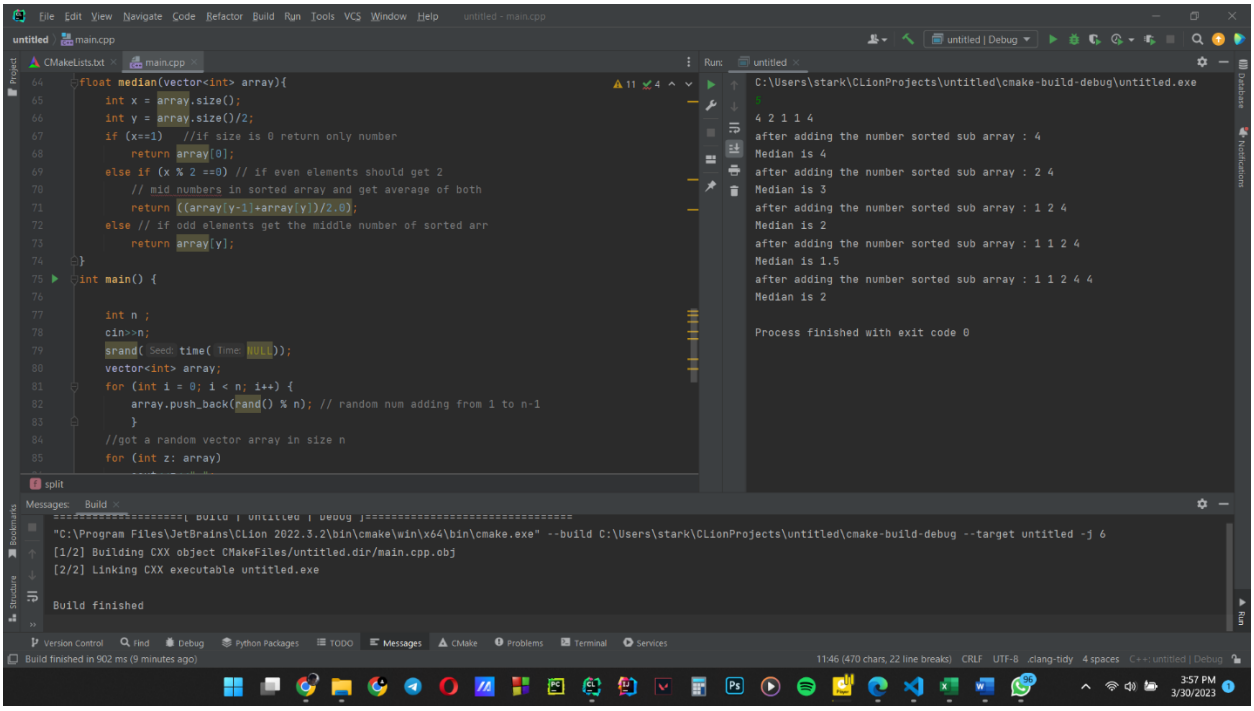
```
int main() {

    int n ;
    cin>>n;
    srand(time(NULL));
    vector<int> array;
    for (int i = 0; i < n; i++) {
        array.push_back(rand() % n); // random num adding from 1 to n-1
        }
    //got a random vector array in size n
```

```
    for (int z: array)
        cout<<z<<" ";
    cout<<endl;
    for( int i = 0 ; i <n ; i++){
        vector<int> sarr(array.begin(),array.begin()+i+1);
        //getting the sub arrays from size 0 to n
        recquicksort(sarr,0,i);      //sorting using quicksort
        cout<<"after adding the number sorted sub array : ";
        for (int z: sarr)
            cout<<z<<" ";
        cout<<endl;
        cout<<"Median is "<<median(sarr)<<endl;
    }
```

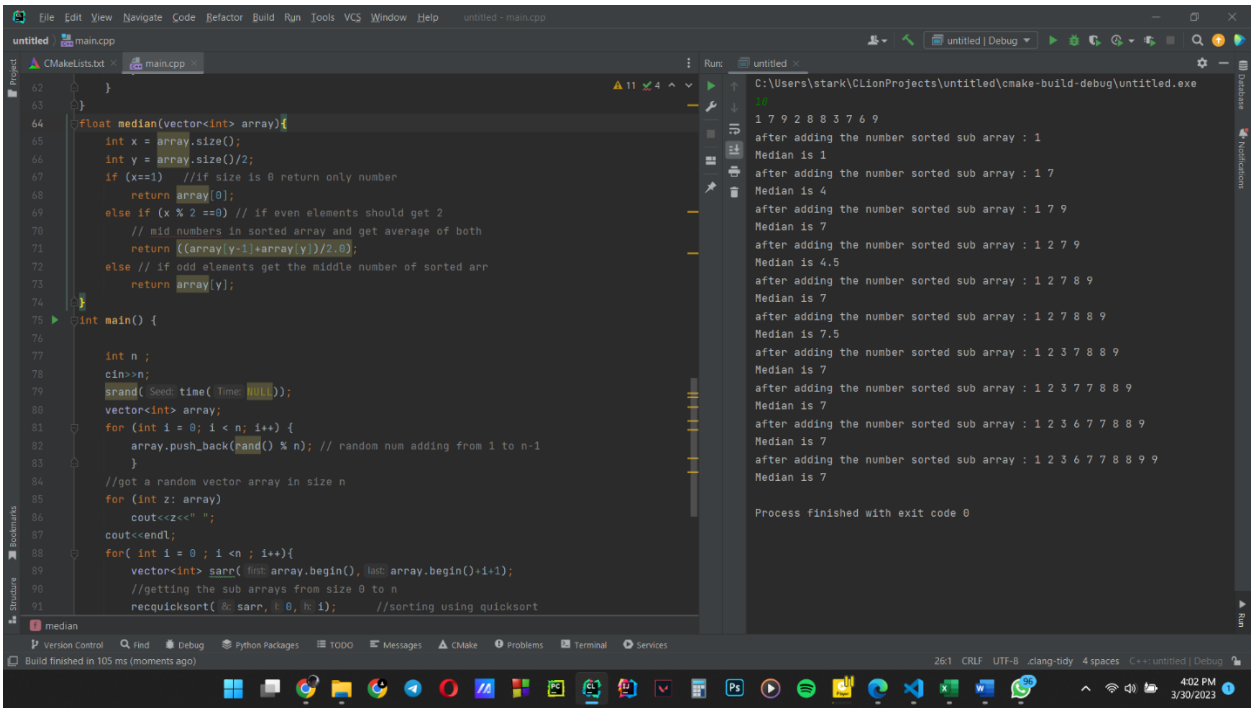
```
}
```

Console outputs examples

For size 5



For size 10


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled - main.cpp
untitled | Debug
CMakeLists.txt
main.cpp
}
}
float median(vector<int> array){
int x = array.size();
int y = array.size()/2;
if (x==1) //if size is 0 return only number
return array[0];
else if (x % 2 ==0) // if even elements should get 2
// mid numbers in sorted array and get average of both
return ((array[y-1]+array[y])/2.0);
else // if odd elements get the middle number of sorted arr
return array[y];
}
int main() {
int n ;
cin>>n;
srand(Seed: time(Time: NULL));
vector<int> array;
for (int i = 0; i < n; i++) {
array.push_back(rand() % n); // random num adding from 1 to n-1
}
//got a random vector array in size n
for (int z: array)
cout<<z<<" ";
cout<<endl;
for(int i = 0 ; i <n ; i++){
vector<int> sarr(first: array.begin(), last: array.begin()+i+1);
//getting the sub arrays from size 0 to n
recquicksort(&: sarr, l: 0, h: i); //sorting using quicksort
median
Run: untitled
C:\Users\stark\CLionProjects\untitled\cmake-build-debug\untitled.exe
10
1 7 9 2 8 8 3 7 6 9
after adding the number sorted sub array : 1
Median is 1
after adding the number sorted sub array : 1 7
Median is 4
after adding the number sorted sub array : 1 7 9
Median is 7
after adding the number sorted sub array : 1 2 7 9
Median is 4.5
after adding the number sorted sub array : 1 2 7 8 9
Median is 7
after adding the number sorted sub array : 1 2 7 8 8 9
Median is 7.5
after adding the number sorted sub array : 1 2 3 7 8 8 9
Median is 7
after adding the number sorted sub array : 1 2 3 7 7 8 8 9
Median is 7
after adding the number sorted sub array : 1 2 3 6 7 7 8 8 9
Median is 7
after adding the number sorted sub array : 1 2 3 6 7 7 8 8 9 9
Median is 7
Process finished with exit code 0
Version Control Find Debug Python Packages TODO Messages CMake Problems Terminal Services
Build finished in 105 ms (moments ago)
26:1 CRLF UTF-8 .clang-tidy 4 spaces C++: untitled | Debug
4:02 PM
3/30/2023