

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

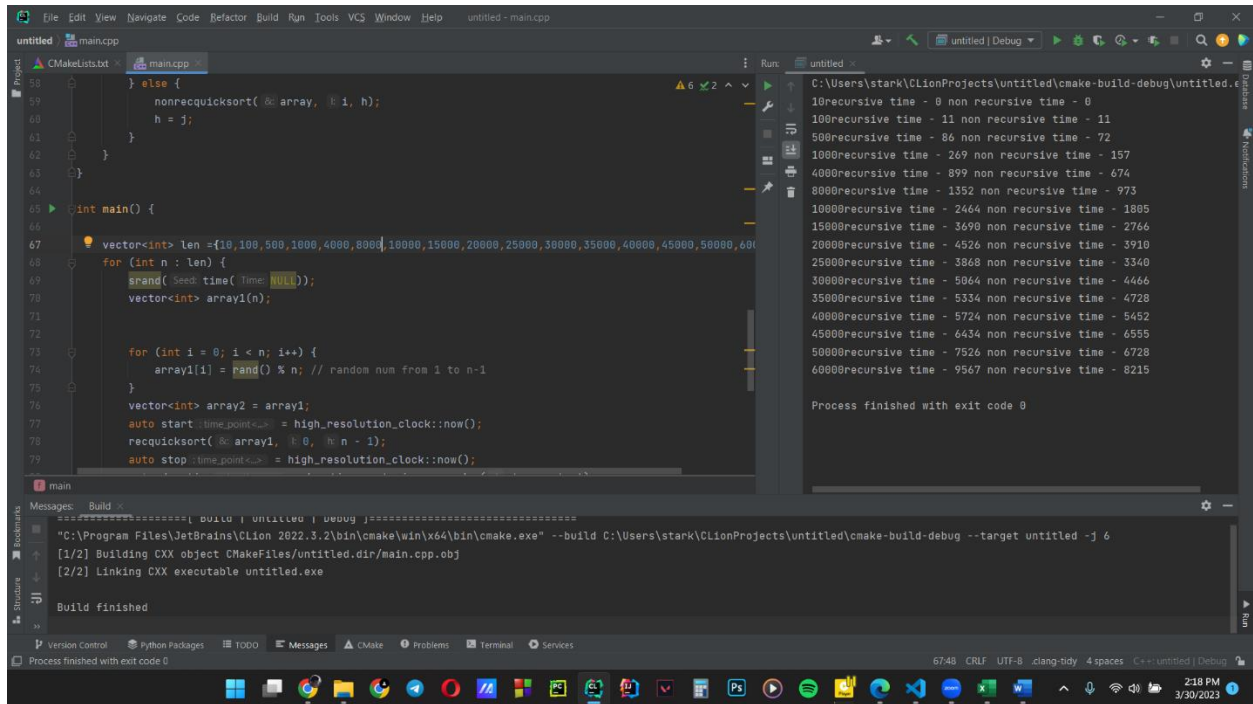
```
void quicksort(vector<int>& array, int l, int h) {
    while (l < h) {
        int i = l, j = h;
        int key = array[(l + h) / 2];

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

Terminal Output:-



The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a recursive quicksort function and a `main` function that tests it with arrays of various sizes. The `main` function uses `high_resolution_clock` to measure the execution time of both recursive and non-recursive versions of the quicksort algorithm. The output of the program is displayed in the console, showing the recursive and non-recursive times for each array size. The build output at the bottom indicates that the program was successfully compiled and linked.

```
58 } else {
59     nonrecursivequickSort(&array, i, h);
60     h = j;
61 }
62 }
63 }
64 }
65 int main() {
66
67     vector<int> len = {10,100,500,1000,4000,8000,10000,15000,20000,25000,30000,35000,40000,45000,50000,60000};
68     for (int n : len) {
69         srand(Seed::time(Time::NULL));
70         vector<int> array1(n);
71
72         for (int i = 0; i < n; i++) {
73             array1[i] = rand() % n; // random num from 1 to n-1
74         }
75         vector<int> array2 = array1;
76         auto start = high_resolution_clock::now();
77         recursivequickSort(&array1, 0, n - 1);
78         auto stop = high_resolution_clock::now();
79
80         cout << "Recursive time - " << (stop - start).count() << " non recursive time - " << (stop - start).count() << endl;
81     }
82 }
```

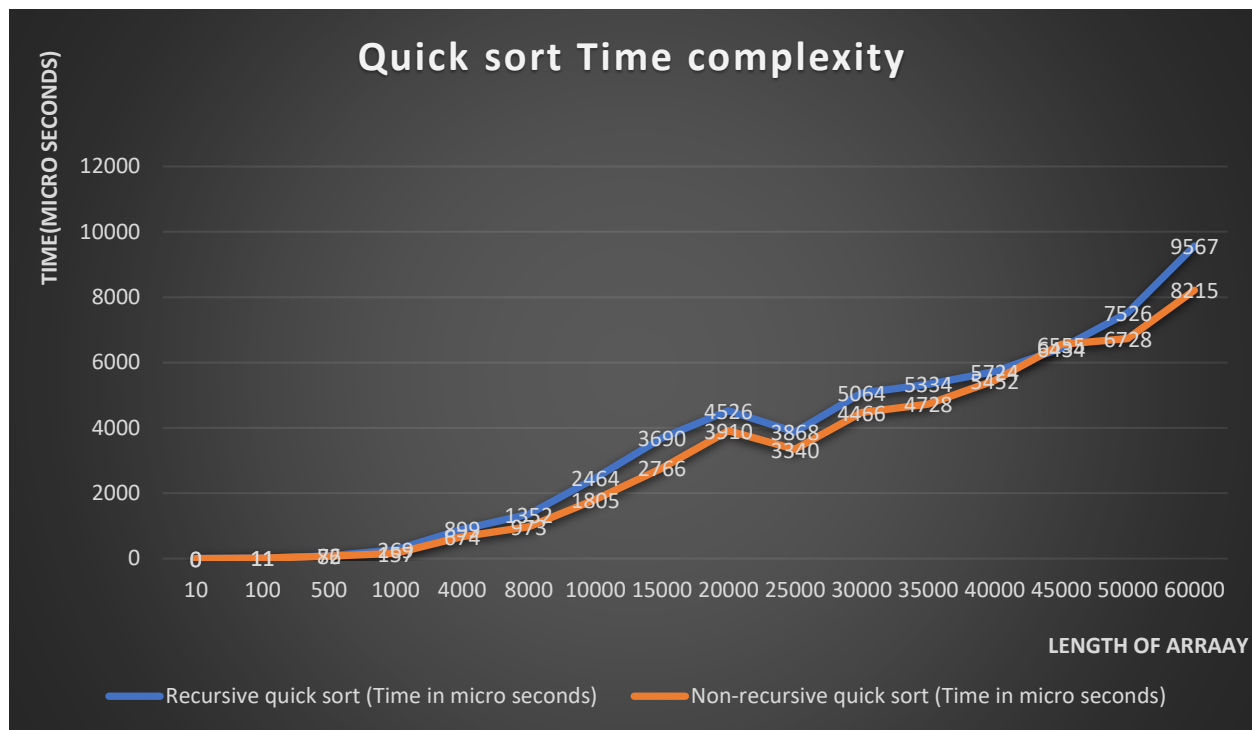
Process finished with exit code 0

Build finished

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 $O(n \log n)$.
- 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

The screenshot shows a C++ IDE with a project named 'untitled'. The code in 'main.cpp' defines a function to find the median of a vector and a main function that generates random subarrays of size 5 and prints their medians. The console output shows the results for several subarrays.

```

//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(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 Output:

```

C:\Users\stark\CLionProjects\untitled\cmake-build-debug\untitled.exe
4 2 1 1 4
after adding the number sorted sub array : 4
Median is 4
after adding the number sorted sub array : 2 4
Median is 3
after adding the number sorted sub array : 1 2 4
Median is 2
after adding the number sorted sub array : 1 1 2 4
Median is 1.5
after adding the number sorted sub array : 1 1 2 4 4
Median is 2
Process finished with exit code 0

```

Build Messages:

```

===== build : untitled : debug =====
"C:\Program Files\JetBrains\CLion 2022.3.2\bin\cmake\win\x64\bin\cmake.exe" --build C:\Users\stark\CLionProjects\untitled\cmake-build-debug --target untitled -j 6
[1/2] Building CXX object CMakeFiles/untitled.dir/main.cpp.obj
[2/2] Linking CXX executable untitled.exe
Build finished

```


For size 10

The screenshot shows the CLion IDE with a C++ project named 'untitled'. The main.cpp file contains the following code:

```
62 }
63
64 float median(vector<int> array){
65     int x = array.size();
66     int y = array.size()/2;
67     if (x==1) //if size is 0 return only number
68         return array[0];
69     else if (x % 2 ==0) // if even elements should get 2
70         // mid numbers in sorted array and get average of both
71         return ((array[y-1]+array[y])/2.0);
72     else // if odd elements get the middle number of sorted arr
73         return array[y];
74 }
75
76 int main() {
77     int n ;
78     cin>>n;
79     srand( Seed: time( NULL));
80     vector<int> array;
81     for (int i = 0; i < n; i++) {
82         array.push_back(rand() % n); // random num adding from 1 to n-1
83     }
84     //got a random vector array in size n
85     for (int z: array)
86         cout<<z<<" ";
87     cout<<endl;
88     for (int i = 0; i < n; i++){
89         vector<int> sarr( first array.begin(), last array.begin()+i+1);
90         //getting the sub arrays from size 0 to n
91         recquicksort( &sarr, 0, i); //sorting using quicksort
92     }
```

The Run window shows the output of the program for n=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 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
```