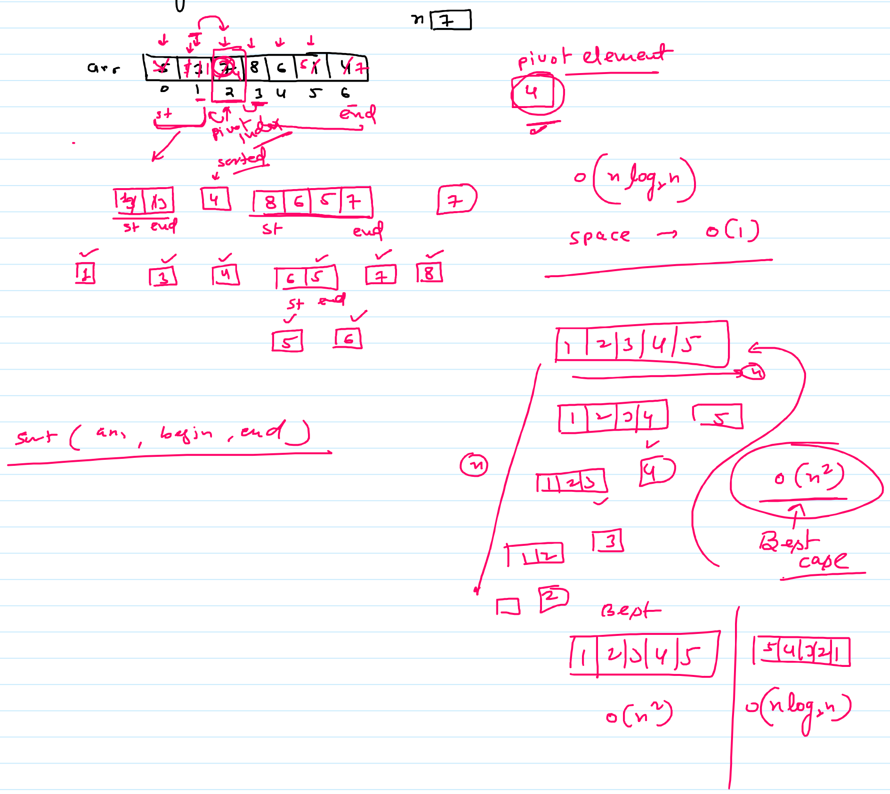
Day 14

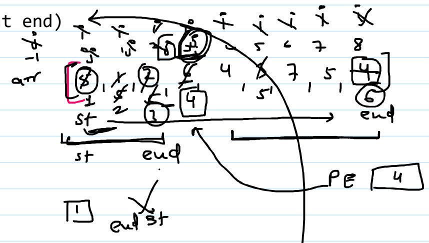
02 April 2025 08:30 AM

Quick Sorting :-

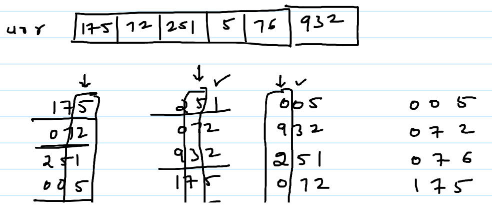


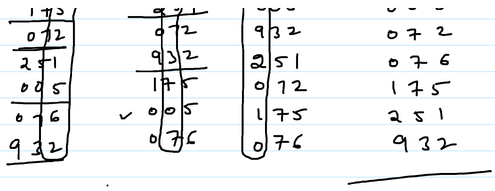
```
#include<iostream>
using namespace std;
int partition(int arr[], int st, int end)
{
    int i,j;
    int pivot_element = arr[end];
    for(i=st,j=st-1 ; i<end ; i++)
    {
        if(arr[i] < pivot_element)
        {
            j++;
            swap(arr[i],arr[j]);
        }
    }
    j++;
    swap(arr[j],arr[end]);
    return j;
}
void quickSort(int arr[], int st, int end)
{
    if(st>=end)
        return;
    int pivot_index = partition(arr,st,end);
    quickSort(arr,st,pivot_index-1);
    quickSort(arr,pivot_index+1,end);
}
int main()
{
    int arr[] = {5,1,2,6,4,3,7,5,4};
    int n = sizeof(arr)/sizeof(int);
    quickSort(arr,0,n-1);

    for(int i:arr)
        cout<<i<<" ";
    return 0;
}
```

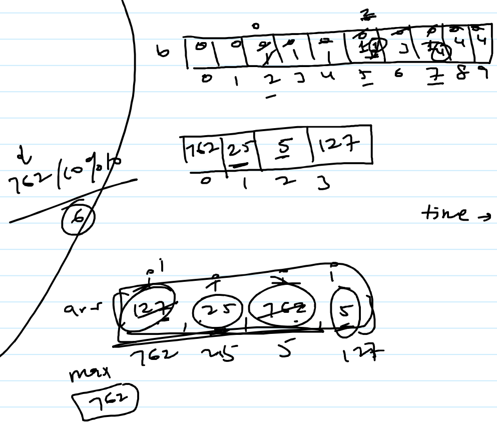


Radix ~~Shell~~ sorting =>





```
#include<iostream>
using namespace std;
void countingSort(int arr[], int n, int place)
{                                               10
    int b[10]={0};
    //count frequency
    for(int i=0;i<n;i++)
    {
        b[arr[i]/place%10]++;
    }
    //left sum   1%10
    for(int i=1;i<10;i++)
        b[i] += b[i-1];
    //match index
    int c[n];
    for(int i=n-1;i>=0;i--)
    {
        c[b[arr[i]/place%10]-1]=arr[i];
        b[arr[i]/place%10]--;
    }
    //copy from c to arr
    for(int i=0;i<n;i++)
        arr[i] = c[i];
}
          radix
void ~~shell~~Sorting(int arr[], int n)
{
    int max = arr[0];
    for(int i=1 ; i<n ; i++)
        if(arr[i]>max)
            max=arr[i];
    for(int i=1; i<=max; i=i*10)
        counting~~Sort~~(arr,n,i);
}
int main()
{
    int arr[] = {257,5,78,61,458,963,2,54};
    int n = sizeof(arr)/sizeof(int);
    ~~shell~~Sorting(arr,n);
    radix
    for(int i:arr)
        cout<<i<<" ";
    return 0;
}
```



time → O(dn)
↑
num of digits (place)

STL → set | multi set | unordered_set | unordered_multiset

set → Collection of unique element | searching time O(log₂n)

→ sorted

include <set>

object
↓
set<int> s;

s.find(num)
found → &num (iterator)
not found → & last +1
end()

```
#include<iostream>
#include<set>
using namespace std;
void output(set<int> &s)
{
    for(auto i:s)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    set<int> s1;
    s1.insert(5);
    s1.insert(2);
    s1.insert(8);
    s1.insert(5);
    s1.insert(3);
    s1.insert(6);
    output(s1);

    auto i = s1.find(5);
    // if(i!=s1.end())
    //     cout<<"found";
    // else
```

```
#include<iostream>
#include<set>
using namespace std;
void output(set<int,greater<int>> &s)
{
    for(auto i:s)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    set<int,greater<int>> s1;
    s1.insert(5);
    s1.insert(2);
    s1.insert(8);
    s1.insert(5);
    s1.insert(3);
    s1.insert(6);
    output(s1);
    auto i = s1.find(2);
    if(i!=s1.end())
        cout<<"Found";
    else
        cout<<"Not found";

    return 0;
}
```

```
    //     cout<<"Not Found";
    if(i!=s1.end())
        s1.erase(i);
    output(s1);

    return 0;
}
```

multi set → Duplicate allow

→ sorted

```
#include<iostream>
#include<set>
using namespace std;
void output(multiset<int> &s)
{
    for(auto i:s)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    multiset<int> s1;
    s1.insert(5);
    s1.insert(2);
    s1.insert(8);
    s1.insert(5);
    s1.insert(3);
    s1.insert(6);
    output(s1);
    auto i = s1.find(60);
    if(i!=s1.end())
        cout<<"Found";
    else
        cout<<"Not found";

    return 0;
}
```

unordered_set :- unique element

→ ~~sorted~~

searching speed

O(1)

↑

Hashing

```
#include<iostream>
#include<unordered_set>
using namespace std;
void output(unordered_set<int> &s)
{
    for(auto i:s)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    unordered_set<int> s1;
    s1.insert(5);
    s1.insert(2);
    s1.insert(8);
    s1.insert(5);
    s1.insert(3);
    s1.insert(6);
    output(s1);
    auto i = s1.find(2);
    if(i!=s1.end())
        cout<<"Found";
    else
        cout<<"Not found";

    return 0;
}
```

unordered_multiset → duplicate accept

→ ~~sorted~~

```
#include<iostream>
#include<unordered_set>
using namespace std;
void output(unordered_multiset<int> &s)
{
    for(auto i:s)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    unordered_multiset<int> s1;
    s1.insert(5);
    s1.insert(2);
    s1.insert(8);
```

```
        s1.insert(5);
        s1.insert(3);
        s1.insert(6);
        output(s1);
        auto i = s1.find(2);
        if(i!=s1.end())
            cout<<"Found";
        else
            cout<<"Not found";

        return 0;
    }
```

map

pair

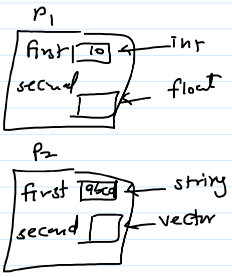
class pair
{ public:
T1 first
T2 second

pair<int, float> p1

pair<string, vector<int>> p2

p1.first = 10;

p2.first = "abcd";



```
#include<iostream>
#include<vector>
using namespace std;
void output(pair<string,vector<int>> &p2)
{
    cout<<p2.first<<" ";
    for(int i:p2.second)
        cout<<i<<" ";
    cout<<endl;
}
int main()
{
    pair<int,float> p1;
    p1.first=10;
    p1.second = 7.5;
    cout<<p1.first<<" "<<p1.second<<endl;
    pair<string,vector<int>> p2;
    p2.first="abcd";
    p2.second.push_back(10);
    p2.second.push_back(20);
    p2.second.push_back(30);
    p2.second.push_back(40);
    output(p2);
    return 0;
}
```

map | multimap | unordered_map | unordered_multimap

map → key must be unique

↓ sort According to Key

#include<map>

roll, name

map<int, string> m1;

m1.insert(pair<int, string>(101, "Amit"));

m1.insert(pair<int, string>(102, "Sumit"));

✗ m1.insert(pair<int, string>(101, "Gopal")); ← duplicate

m1[104] = "Rahul";

m1[105] = " Amit"; ✓

m1[105] = " Sureph"; → value Replace

key, value

int	string
105	~~Amit~~ Suresh
104	Rahul
102	sumit
101	Amit

m1

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

```
void output(map<int,string> &m)
{
    for(auto i:m)
    {
        cout<<i.first<<" : "<<i.second<<endl;
    }
    cout<<endl;
}
int main()
{
    map<int,string> m1;
    m1.insert(pair<int,string>(101,"Amit"));
    m1.insert(pair<int,string>(105,"Gopal"));
    m1.insert(pair<int,string>(102,"Gopal"));
    m1.insert(pair<int,string>(102,"Gopal"));
    m1.insert(pair<int,string>(101,"Gopal"));
    m1[103]="Tarun";
    m1[101]="Jatin";
    output(m1);
    auto i = m1.find(105);
    if(i != m1.end())
        cout<<"Found Value = "<<i->second;
    else
        cout<<"Not found";
    return 0;
}
```

```
class map
{
        pair < T1 , T2 >  * p ;
```