Methods Of Sorting lecture-03

Raveesh Gupta

March 2019

Chapter 1

Introduction to Sorting Algorithms

Its the way of arranging the data so it becomes increasing type, Increasing is different for different data types. like sort([2,5,1,2,1]) = [1,1,2,2,5] (non- decreasing)

Types of Sorting Algorithms N - be the size of the data we are trying to sort.
1) Quadratic Sorting algorithms. $O(N^2) :=$ make about N^2 basic operations. N might be around 1E4
2)Linear-log algorithms. $O(N * log(N))$ time N might be around 3E6 - 5E6

For General case there is no linear time sorting algorithm. .

Chapter 2

Quadratic Algorithms

Types of 3 sorting algorithms

2.1 Bubble sort

Bubble sort:-[-7, 6,3 15, 270, 0] -¿ [-7,0,3,6,15,270]
Phase: Comparing adjacent elements and if its in decreasing order we swap them.
After 1st phase:- [-7, 3, 6, 15, 0, 270]
Its called a Bubble sort because if we have a very large data in the beginning it goes to the last at the end of a phase. It turns out that we have to $N-1$ phases. After 1st phase of the bubble sort the largest element in arr[1...N] will be at Nth position. At the ith phase of the bubble sort the ith largest element at the $N-i+1$ position.

```
Input:−
6
−7 6 3 15 270 0
int main(){
    int n,;
    vector<int> a;
    cin>> n;
    for(int i = 0;i<n;i++){
        int x;
        cin>>x;
        a.push_back(x);
    }
    cout<< ”Before:”<< endl;
    for(int i = 0;i<n; i++) printf(”%d ”, a[i]);
```

```
        printf("\n");
        for(int i =0;i< n;i++){
            //(0, 1), (1, 2), ....... (n-2, n-1)
            for(int j = 0; j< n-1; j++)
                if(a[j] > a[j+1])
                    swap(a[j], a[j+1]);
            cout<<"After phase "<<i;
            for(int j = 0;j<n; j++) printf("%d ", a[j]);
            printf("\n");
        }
    }
```

We have $N * (N - 1)$ comparisons which make around N^2

2.2 Insertion Sort

Its the fastest quadratic sorting algorithms.
Consider the first element -7.// its an sorted array.
Consider the second element appended [-7, 6] //its an sorted array
Consider the third element appended [-7, 6, 3] // its not an sorted array
it will move to the left till it finds a non decreasing position. it becomes [-7, 3, 6]
.
.
.

Consider the *ith* element appended -7, 3, 6 .... a_i

if $a_{i-1} > a_i$ the our pointer j moves to the left till it finds $a_{j-1} < a_j$

```
for(int i = 0;i<n;i++){
    int j = i;
    while(j>0 && a[j] < a[j-1]){
        swap(a[j], a[j-1]);
        j--;
    }
}
```

2.3 Selection Sort

We find the minimum number and put it on the *ist* position and now we forge about the first element and and we find the minimum number again in $a[2...n]$ and put it in the $2nd$ position and again forget the prefix and repeat the process for the suffix.

```
for(int i =0; i< n;i++){
    int min_p = -1;
    int mi_v;
    for(int j =i; j< n; j++){
        if(min_p == -1|| a[j]< min_v){
            min_p = j;
            min_v = a[j];
        }
        swap(a[i], a[min_p]);
    }
}
```

Chapter 3

Standard Library C++ $std : sort() : -(pair, struct, int)$

3.1 std:pairs

Sorting of std:pairs done by $std : sort()$

```
std::pair<int, int> my_pair = {2, 5};

int main(){
    int n;
    vector<pair<int, int>> a;
    cin >> n;
    for(int i =0; i < n; i++){
        int x, y;
        cin>>x>> y;
        a.push_back({x, y});
    }
    pair<int, int> my_pair = {11, 3};
    cout<< my_pair.first <<” ”<< my_pair.second<<endl;
    return 0;
}
```

Suppose we have a list of scores:

```
5
a 100
b 90
c 115
d 100
e 0
```

```
std:pair<int, int> my_pair = {2, 5};

int main(){
    int n;
    vector<pair<int, int>> a;
    cin >> n;
    for(int i =0; i < n; i++){
        string name; int score;
        cin>>name>> score;
        a.push_back({score, name});
    }
    sort(a.begin(), a.end());
    for(int  i =0; i<n; i++)
        cout<< a[i].first << " "<< a[i].second<<endl;
    return 0;
}
```

Comparison of $std:pairs$ of strings in lexicographical order .

```
p1 < p2
if a_1 != a_2;
    return a_1< a_2
if b_1 != b_2
    return b_1<b_2
return 0 // p1==p2
```

```
    std:pair<int, int> my_pair = {2, 5};
    bool cmp(pair<int, string> &a, pair<int, string> &b){
    //return true if a < b
    //return false if a >= b
        if(a.first != b.first)
            return a.first > b.first;
        return a.second < b.second;
    }
    bool cmp_1(pair<int, string> &a, pair<int, string> &b){
        if(a.first != b.first)
            return a.first > b.first;
        if (a.second.size() != b.second().size())
    }
    int main(){
        int n;
        vector<pair<int, int>> a;
        cin >> n;
        for(int i =0; i < n; i++){
            string name; int score;
            cin>>name>> score;
            a.push_back({score, name});
        }
        sort(a.begin(), a.end(), cmp);
        for(int i =0; i<n; i++)
            cout<< a[i].first << ” ”<< a[i].second<<endl;
        return 0;
    }
```

Chapter 4

Greedy Algorithms

Greedy algorithms tells us whats best for know and don't think of the future.

4.1 Problem: Number line

We have some n segments (pairs) on a number line. Least number of segments that cover all the segments on the number line.