Design and Analysis of Algorithms

Shivanshu

December 14, 2021

Index

1 Week 1

1.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int key;
        cin >> key;
        int count = 0;
        bool flag = false;
        for (int i = 0; i < v.size(); i++) {
            if (++count && v[i] == key) {
                flag = true;
                break;
            }
        }
        if (flag) {
            cout << "Present " << count << endl;
        } else {
            cout << "Not Present " << count << endl;
        }
    }
    return 0;
}
```

Input

```
3
8
34 35 65 31 25 89 64 30
89
5
977 354 244 546 355
244
6
23 64 13 67 43 56
63
```

Output

```
Present 6
Present 3
Not Present 6
```

1.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }

        int key;
        cin >> key;

        int start = 0, end = n - 1;
        int mid = (start + end) / 2;

        bool flag = false;
        int count = 0;
        while (start <= end) {
            mid = (start + end) / 2;
            if (++count && v[mid] == key) {
                flag = true;
                break;
            } else if (v[mid] < key) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        if (flag) {
            cout << "Present " << count << "\n";
        } else {
            cout << "Not Present " << count << "\n";
        }
    }
    return 0;
}
```

Input

```
3
5
12 23 36 39 41
41
8
21 39 40 45 51 54 68 72
69
10
101 246 438 561 796 896 899 4644 7999 8545
7999
```

Output

```
Present 3
Not Present 4
Present 3
```

1.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int key;
        cin >> key;
        bool flag = false;
        int count = 0;
        int power = 0;
        while (++count && power < n && key >= v[power]) {
            if (key == v[power]) {
                flag = true;
                break;
            }
            power = (power == 0) ? 2 : power * 2;
        }
        if (!flag) {
            int start = power / 2;
            int end = min(power, n);
            if (end == 0) {
                flag = false;
            } else {
                for (int i = start + 1; i < end; i++) {
                    if (++count && v[i] == key) {
                        flag = true;
                        break;
                    }
                }
            }
        }
        if (flag) {
            cout << "Present " << count << "\n";
        } else {
            cout << "Not Present " << count << "\n";
        }
    }
    return 0;
}
```

Input

```
3
5
12 23 36 39 41
41
8
21 39 40 45 51 54 68 72
69
10
101 246 438 561 796 896 899 4644 7999 8545
7999
```

Output

```
Present 3
Not Present 7
Present 4
```

2 Week 2

2.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int key;
        cin >> key;
        int lb = 0, ub = 0;
        int start = 0, end = n - 1;
        while (start < end) {
            int mid = (start + end) / 2;
            if (v[mid] == key) {
                lb = mid;
                ub = mid + 1;
                for (int j = mid - 1; j >= 0 && v[j] == key; j--) lb--;
                for (int j = mid + 1; j < n && v[j] == key; j++) ub++;
                break;
            } else if (v[mid] < key) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        if (ub - lb > 0) {
            cout << key << " - " << ub - lb << "\n";
        }
    }
    return 0;
}
```

Input

```
2
10
235 235 278 278 763 764 790 853 981 981
981
15
1 2 2 3 3 5 5 5 25 75 75 75 97 97 97
75
```

Output

```
981 - 2
75 - 3
```

2.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        vector<int> ans;
        bool flag = false;
        for (int i = 0; i < n - 2; i++) {
            for (int j = i + 1; j < n - 1; j++) {
                // element to search in the location j+1 to n-1
                int key = v[i] + v[j];
                // using binary serach
                int start = j + 1, end = n - 1;
                while (start <= end) {
                    int mid = (start + end) / 2;
                    if (v[mid] == key) {
                        ans.push_back(i);
                        ans.push_back(j);
                        ans.push_back(mid);
                        flag = true;
                        break;
                    } else if (v[mid] < key) {
                        start = mid + 1;
                    } else {
                        end = mid - 1;
                    }
                }
                if (flag) break;
            }
            if (flag) break;
        }
        // printing result
        if (flag) {
            cout << ans[0]+1 << "," << ans[1]+1 << "," << ans[2]+1 << "\n";
        } else {
            cout << "No sequence found.\n";
        }
    }
    return 0;
}
```

Input

```
3
5
1 5 84 209 341
10
24 28 48 71 86 89 92 120 194 201
15
64 69 82 95 99 107 113 141 171 350 369 400 511 590 666
```

Output

```
No sequence found.
2,7,8
1,6,9
```

2.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int key;
        cin >> key;
        sort(v.begin(), v.end());
        int count = 0;
        for (int i = 0; i < n - 1; i++) {
            int find = key + v[i];
            int start = i + 1, end = n - 1;
            while (start <= end) {
                int mid = (start + end) / 2;
                if (v[mid] == find) {
                    count++;
                    break;
                } else if (v[mid] < find) {
                    start = mid + 1;
                } else {
                    end = mid - 1;
                }
            }
        }
        cout << count << "\n";
    }
    return 0;
}
```

Input

```
2
5
1 51 84 21 31
20
10
24 71 16 92 12 28 48 14 20 22
4
```

Output

```
2
4
```

3 Week 3

3.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int comp = 0, shifts = 0;
        for (int i = 1; i < n; i++) {
            int key = v[i];
            int j = i - 1;
            while (++comp && j >= 0 && key < v[j]) {
                v[j + 1] = v[j];
                shifts++;
                j--;
            }
            j++;
            v[j] = key;
        }
        for (int i = 0; i < n; i++) {
            cout << v[i] << " ";
        }
        cout << endl;
        cout << "comparisions = " << comp << "\n";
        cout << "shifts = " << shifts << "\n";
    }
    return 0;
}
```

Input

```
3
8
-23 65 -31 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 -12 54 65 86 46 325
```

Output

```
-31 -23 32 45 46 65 76 89
comparisions = 20
shifts = 13
21 32 34 46 51 54 65 76 78 97
comparisions = 37
shifts = 28
-12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisions = 68
shifts = 54
```

3.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        int comp = 0, swap = 0;
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (comp++ && v[min] > v[j]) {
                    min = j;
                }
            }
            if (i != min) {
                int temp = v[min];
                v[min] = v[i];
                v[i] = temp;
                swap++;
            }
        }
        for (int i = 0; i < n; i++) {
            cout << v[i] << " ";
        }
        cout << "\n";
        cout << "comparisions = " << comp << "\n";
        cout << "swaps = " << swap << "\n";
    }
    return 0;
}
```

Input

```
3
8
-13 65 -21 76 46 89 45 12
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```

Output

```
-21 -13 12 45 46 65 76 89
comparisions = 28
swaps = 5
21 32 34 46 51 54 65 76 78 97
comparisions = 45
swaps = 6
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisions = 105
swaps = 12
```

3.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> v(n);
        for (int i = 0; i < n; i++) {
            cin >> v[i];
        }
        bool flag = true;
        for (int i = 1; i < n; i++) {
            int key = v[i];
            int j = i - 1;
            while (j >= 0 && key < v[j]) {
                v[j + 1] = v[j];
                j--;
            }
            j++;
            v[j] = key;
            if (v[j - 1] == v[j]) {
                flag = false;
                break;
            }
        }
        if (!flag) {
            cout << "Yes\n";
        } else {
            cout << "No\n";
        }
    }
    return 0;
}
```

Input

```
3
5
28 52 83 14 75
10
75 65 1 65 2 6 86 2 75 8
15
75 35 86 57 98 23 73 1 64 8 11 90 61 19 20
```

Output

```
No
Yes
No
```

4 Week 4

4.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
vector<int> merge(vector<int> left, vector<int> right, int *comp, int *inv) {
    vector<int> merged(left.size() + right.size());
    int i, j, k;
    for (i = 0, j = 0, k = 0; i < left.size() && j < right.size();) {
        if (left[i] < right[j]) {
            merged[k++] = left[i++];
        } else {
            merged[k++] = right[j++];
            (*inv) = (*inv) + left.size() - i;
        }
        (*comp)++;
    }
    while (i < left.size()) merged[k++] = left[i++];
    while (j < right.size()) merged[k++] = right[j++];
    return merged;
}

vector<int> merge_sort(vector<int> v, int *comp, int *inv) {
    if (v.size() == 1) {
        return v;
    }
    int mid = v.size() / 2;
    vector<int> left(mid);
    vector<int> right(v.size() - mid);
    for (int i = 0; i < mid; i++) left[i] = v[i];
    for (int i = mid; i < v.size(); i++) right[i - mid] = v[i];
    vector<int> left2 = merge_sort(left, comp, inv);
    vector<int> right2 = merge_sort(right, comp, inv);
    vector<int> ans = merge(left2, right2, comp, inv);
    return ans;
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }
        int comparisions=0, inversions=0;
        vector<int> sarr = merge_sort(arr, &comparisions, &inversions);
        for (int i = 0; i < sarr.size(); i++) {
            cout << sarr[i] << " ";
        }
```

```
        cout << "\n";
        cout << "comparisions = " << comparisions << "\n";
        cout << "inversions = " << inversions << "\n";
    }
    return 0;
}
```

Input

```
3
8
23 65 21 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```

Output

```
21 23 32 45 46 65 76 89
comparisions = 16
inversions = 13
21 32 34 46 51 54 65 76 78 97
comparisions = 23
inversions = 28
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisions = 41
inversions = 54
```

4.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int partition(vector<int> &arr, int start, int end, int &comp, int &swap) {
    int pivot = arr[end]; // end element is pivot
    int i = start - 1;    // right postiton of pivot
    for (int j = start; j < end; j++) {
        if (++comp && arr[j] < pivot) {
            i++;
            swap++;
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
        }
    }
    if (i + 1 != end) {
        swap++;
        int temp = arr[i + 1];
        arr[i + 1] = arr[end];
        arr[end] = temp;
    }
    return i + 1;
}
// random pivot
int partition_r(vector<int> &arr, int l, int r, int &comp, int &swap) {
    srand(time(NULL));
    int random = l + rand() % (r - l);
    swap++;
    int temp = arr[random];
    arr[random] = arr[r];
    arr[r] = temp;

    return partition(arr, l, r, comp, swap);
}

/* quick sort algorithm */
void quick_sort(vector<int> &arr, int start, int end, int &comp, int &swap) {
    if (start < end) {
        int pivot = partition_r(arr, start, end, comp, swap);
        quick_sort(arr, start, pivot - 1, comp, swap);
        quick_sort(arr, pivot + 1, end, comp, swap);
    }
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
```

```
            cin >> arr[i];
        }
        int comparisions = 0, swaps = 0;
        quick_sort(arr, 0, arr.size() - 1, comparisions, swaps);
        for (int i = 0; i < arr.size(); i++) cout << arr[i] << " ";
        cout << "\n";
        cout << "comparisions: " << comparisions << "\n";
        cout << "swaps: " << swaps;
        cout << "\n";
    }
    return 0;
}
```

Input

```
3
8
23 65 21 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```

Output

```
21 23 32 45 46 65 76 89
comparisions: 18
swaps: 13
21 32 34 46 51 54 65 76 78 97
comparisions: 23
swaps: 20
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisions: 43
swaps: 42
```

4.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;

// partition form the end
int partitionSmall(int arr[], int l, int r) {
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++) {
        if (arr[j] <= x) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
        }
    }
    int temp = arr[i];
    arr[i] = arr[r];
    arr[r] = temp;
    return i;
}

/* quick select algortihm for smallest k */
int quickSelectSmall(int arr[], int l, int r, int k) {
    if (k > 0 && k <= r - l + 1) {
        // partition the array around the last element
        int index = partitionSmall(arr, l, r);
        // if index is same as k
        if (index - l == k - 1) {
            return arr[index];
        }
        // if index is more, recurse for left, else right
        if (index - l > k - 1) {
            return quickSelectSmall(arr, l, index - 1, k);
        } else {
            return quickSelectSmall(arr, index + 1, r, k - index + l - 1);
        }
    }
    return INT_MIN;
}

// partition form the end
int partitionLarge(int arr[], int l, int r) {
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++) {
        if (arr[j] >= x) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
        }
```

```
    }
    int temp = arr[i];
    arr[i] = arr[r];
    arr[r] = temp;
    return i;
}

/* quick select algorithm for largest k */
int quickSelectLarge(int arr[], int l, int r, int k) {
    if (k > 0 && k <= r - l + 1) {
        // partition the array around the last element
        int index = partitionLarge(arr, l, r);
        // if index is same as k
        if (index - l == k - 1) {
            return arr[index];
        }
        // if index is more, recurse for left, else right
        if (index - l > k - 1) {
            return quickSelectLarge(arr, l, index - 1, k);
        } else {
            return quickSelectLarge(arr, index + 1, r, k - index + l - 1);
        }
    }
    return INT_MAX;
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }
        int k;
        cin >> k;
        // find the kth smallest and largest element
        if (k > n) {
            cerr << "Invalid input\n";
            continue;
        }
        int smallk = quickSelectSmall(arr, 0, n - 1, k);
        int largek = quickSelectLarge(arr, 0, n - 1, k);
        cout << smallk << "\n"
             << largek << "\n";
    }
    return 0;
}
```

Input

```
2
10
123 656 54 765 344 514 765 34 765 234
3
15
43 64 13 78 864 346 786 456 21 19 8 434 76 270 601
8
```

Output

```
123
765
78
78
```

5 Week 5

5.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    /* no of test case */
    int t;
    cin >> t;
    while (t--) {
        /* length of array */
        int n;
        cin >> n;
        /* array to store */
        char a[n];
        int count[27] = {0};
        for (int i = 0; i < n; i++) {
            cin >> a[i];
            count[a[i] - 'a']++;
        }
        /* finding the maximum count index */
        int max_i = 0;
        for (int i = 0; i < 27; i++) {
            if (count[i] > count[max_i]) {
                max_i = i;
            }
        }
        if (count[max_i] == 1 || count[max_i] == 0) {
            cout << "No Duplicates Present\n";
        } else {
            cout << (char)(max_i + 'a') << "-" << count[max_i] << "\n";
        }
    }
    return 0;
}
```

Input

```
3
10
a e d w a d q a f p
15
r k p g v y u m q a d j c z e
20
g t l l t c w a w g l c w d s a a v c l
```

Output

```
a-3
No Duplicates Present
l-4
```

5.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    /* test cases */
    int t;
    cin >> t;
    while (t--) {
        /* array input */
        int n;
        cin >> n;
        int arr[n] = {0};
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }
        int key;
        cin >> key;
        /* creating hashmap for the elements */
        unordered_map<int, int> m;
        bool found = false;
        for (int i = 0; i < n; i++) {
            /* element found */
            if (m.find(key - arr[i]) != m.end()) {
                found = true;
                cout << arr[i] << " " << (key - arr[i]) << ", ";
            }
            /* register it's presence */
            m[arr[i]]++;
        }
        if (!found) {
            cout << "No Such Pair Exist\n";
        } else {
            cout << "\n";
        }
    }
    return 0;
}
```

Input

```
2
10
64 28 97 40 12 72 84 24 38 10
50
15
56 10 72 91 29 3 41 45 61 20 11 39 9 12 94
302
```

Output

```
38 12, 10 40,
No Such Pair Exist
```

5.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    /* first array */
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++) cin >> a[i];
    /* second array */
    int m;
    cin >> m;
    int b[m];
    for (int i = 0; i < m; i++) cin >> b[i];
    int i = 0, j = 0;
    while (i < n && j < m) {
        if (a[i] == b[j]) {
            cout << a[i] << " ";
            i++;
            j++;
        } else if (a[i] < b[j]) {
            i++;
        } else {
            j++;
        }
    }
    return 0;
}
```

Input

```
7
10 10 34 39 55 76 85
12
10 10 11 30 30 34 34 51 55 69 72 89
```

Output

```
10 10 34 55
```

6 Week 6

6.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
void dfs(vector<int> arr[], int source, int V, bool *visited) {
    visited[source] = true;
    for (int i = 0; i < V; i++) {
        if (arr[source][i] != 0 && !visited[i]) {
            // adjacent vertex
            dfs(arr, i, V, visited);
        }
    }
}

bool checkPath(vector<int> arr[], int V, int source, int destination) {
    bool visited[V];
    for (int i = 0; i < V; i++) visited[i] = false;
    dfs(arr, source, V, visited);
    return visited[destination];
}

int main() {
    int n;
    cin >> n;
    vector<int> arr[n];
    /* input the adjacency matrix */
    int temp;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> temp;
            arr[i].push_back(temp);
        }
    }
    int source, destination;
    cin >> source >> destination;
    if (checkPath(arr, n, source - 1, destination - 1)) {
        cout << "Yes Path Exists.\n";
    } else {
        cout << "No Such Path Exists.\n";
    }
    return 0;
}
```

Input

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
1 5
```

Output

```
Yes Path Exists.
```

6.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;

bool isBipartiteUtil(vector<int> G[], int src, int colorArr[], int V) {
    colorArr[src] = 1; // set one color
    queue<int> q;
    q.push(src);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        // self loop
        if (G[u][u] == 1) return false;
        for (int v = 0; v < V; ++v) {
            if (G[u][v] != 0 && colorArr[v] == -1) {
                colorArr[v] = 1 - colorArr[u]; // assign opposite color
                q.push(v);
            } else if (G[u][v] != 0 && colorArr[v] == colorArr[u])
                return false; // edge with same color exist
        }
    }
    return true;
}

bool isBipartite(vector<int> G[], int V) {
    /* -1 -> no color, 1-> red color, 0 -> blue color */
    int colorArr[V];
    for (int i = 0; i < V; ++i) colorArr[i] = -1;
    for (int i = 0; i < V; i++)
        if (colorArr[i] == -1)
            if (isBipartiteUtil(G, i, colorArr, V) == false)
                return false;
    return true;
}

int main() {
    int n;
    cin >> n;
    vector<int> G[n];
    int temp;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> temp;
            G[i].push_back(temp);
        }
    }
    if (isBipartite(G, n)) {
        cout << "Yes Bipartite\n";
    } else {
        cout << "Not Bipartite\n";
    }
    return 0;
```

```
}
```

Input

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
```

Output

```
Not Bipartite
```

7 Week 7

7.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;

int bellman_ford(int **graph, int source, int destination, int vertices) {
    int dist[vertices];
    for (int i = 0; i < vertices; i++) {
        dist[i] = INT_MAX;
    }
    dist[source] = 0;
    for (int i = 0; i < vertices - 1; i++) {
        for (int j = 0; j < vertices; j++) {
            for (int k = 0; k < vertices; k++) {
                if (dist[j] != INT_MAX && graph[j][k] != 0 && dist[j] + graph[
   j][k] < dist[k]) {
                    dist[k] = dist[j] + graph[j][k];
                }
            }
        }
    }
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            if (dist[j] != INT_MAX && graph[j][i] != 0 && dist[j] + graph[j][i
   ] < dist[i]) {
                return -1;
            }
        }
    }
    return dist[destination];
}
int main() {
    int vertices, edges;
    cin >> vertices >> edges;
    int **graph = new int *[vertices];
    for (int i = 0; i < vertices; i++) {
        graph[i] = new int[vertices];
        for (int j = 0; j < vertices; j++) {
            graph[i][j] = 0;
        }
    }
    for (int i = 0; i < edges; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
    }
    int source, destination;
    cin >> source >> destination;
    int ans = bellman_ford(graph, source, destination, vertices);
    if (ans == INT_MAX) {
        cout << "-1";
```

```
    } else {
        cout << ans;
    }
    return 0;
}
```

Input

```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
```

Output

```
-1
```

7.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;

void calulate(vector<int> &pa, int i) {
    cout << i + 1 << " ";
    if (pa[i] >= 0)
        calulate(pa, pa[i]);
}

void find_path(int **graph, int m, int sour) {
    vector<int> dis(m, INT_MAX), pa(m, -1);
    dis[sour] = 0;
    for (int ki = 0; ki < m - 1; ki++) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < m; j++) {
                if (graph[i][j] != 0) {
                    if (dis[j] > dis[i] + graph[i][j]) {
                        dis[j] = dis[i] + graph[i][j];
                        pa[j] = i;
                    }
                }
            }
        }
    }
    for (int i = 0; i < m; i++) {
        calulate(pa, i);
        cout << ": " << dis[i] << endl;
    }
}

int main() {
    int m, source, ed;
    cin >> m;
    int **graph = (int **)malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++)
        graph[i] = (int *)malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            cin >> graph[i][j];
        }
    }
    cin >> source;
    find_path(graph, m, source - 1);
}
```

Input

```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
```

Output

```
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
```

7.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;

int shortest_weigt(int **graph, int ver, int u, int v, int k) {
    if (k <= 0)
        return INT_MAX;
    if (k == 0 && u == v)
        return 0;
    if (k == 1 && graph[u][v] != INT_MAX)
        return graph[u][v];
    int res = INT_MAX;
    for (int i = 0; i < ver; i++) {
        if (graph[u][i] != 0 && u != i && v != i) {
            int recu = shortest_weigt(graph, ver, i, v, k - 1);
            if (recu != INT_MAX)
                res = min(res, graph[u][i] + recu);
        }
    }
    return res;
}

int main() {
    int ver, u, v, k, ans;
    cin >> ver;
    int **graph = (int **)malloc(ver * sizeof(int *));
    for (int i = 0; i < ver; i++)
        graph[i] = (int *)malloc(sizeof(int) * ver);
    for (int i = 0; i < ver; i++)
        for (int j = 0; j < ver; j++)
            cin >> graph[i][j];
    cin >> u >> v >> k;
    ans = shortest_weigt(graph, ver, u - 1, v - 1, k);
    cout << "Weight of shortest path from (" << u
         << "," << v << ") with " << k << " edges :" << ans;
}
```

Input

```
4
0 10 3 2
0 0 0 7
0 0 0 6
0 0 0 0
1 4
2
```

Output

```
Weight of shortest path from (1,4) with 2 edges :9
```

8 Week 8

8.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;

struct Edge {
    int weight;
    int a, b;
    Edge(){};
    Edge(int aa, int bb, int w) {
        weight = w;
        a = aa;
        b = bb;
    }
    bool operator()(Edge &a, Edge &b) {
        return a.weight > b.weight;
    }
};

int primsMST(vector<vector<int>> &g) {
    set<int> tree;
    int n = g.size();
    // Min heap
    priority_queue<Edge, vector<Edge>, Edge> q;
    int mn = INT_MAX;
    Edge mini;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (g[i][j] != 0) {
                if (g[i][j] < mn) {
                    mn = g[i][j];
                    mini.a = i;
                    mini.b = j;
                    mini.weight = mn;
                }
            }
        }
    }
    int ans = 0;
    q.push(mini);
    while (!q.empty()) {
        if (tree.size() == n) break;
        Edge u = q.top();
        q.pop();
        int both = 0;
        if (tree.find(u.a) == tree.end()) {
            for (int i = 0; i < n; i++) {
                if (i == u.a or i == u.b) continue;
                if (g[i][u.a] != 0) {
                    Edge temp(i, u.a, g[i][u.a]);
                    q.push(temp);
```

```
                }
                if (g[u.a][i] != 0) {
                    Edge temp(u.a, i, g[u.a][i]);
                    q.push(temp);
                }
            }
            tree.insert(u.a);
            both++;
        }
        if (tree.find(u.b) == tree.end()) {
            for (int i = 0; i < n; i++) {
                if (i == u.b or i == u.a) continue;
                if (g[i][u.b] != 0) {
                    Edge temp(i, u.b, g[i][u.b]);
                    q.push(temp);
                }
                if (g[u.b][i] != 0) {
                    Edge temp(u.b, i, g[u.b][i]);
                    q.push(temp);
                }
            }
            tree.insert(u.b);
            both++;
        }
        if (both != 0) ans += u.weight;
    }
    return ans;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> g(n + 1, vector<int>(n + 1, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> g[i][j];
        }
    }
    int ans = primsMST(g);
    cout << "Minimum Spanning Weight: ";
    cout << ans << '\n';
    return 0;
}
```

Input

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
```

Output

```
Minimum Spanning Weight: 37
```

8.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;

struct UF {
    vector<int> e;
    UF(int n) { e.assign(n, -1); }
    bool sameSet(int a, int b) { return find(a) == find(b); }
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : e[x] = find(e[x]); }
    bool join(int a, int b) {
        a = find(a);
        b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        e[a] += e[b];
        e[b] = a;
        return true;
    }
};

struct Edge {
    int weight;
    int a, b;
    Edge(int aa, int bb, int w) {
        weight = w;
        a = aa;
        b = bb;
    }
};

int kruskalsMST(vector<vector<int>> &g) {
    int n = g.size();
    UF uf(n + 1);
    vector<Edge> edges;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (g[i][j] == 0) continue;
            Edge temp(i, j, g[i][j]);
            edges.push_back(temp);
        }
    }
    sort(edges.begin(), edges.end(), [&](Edge &a, Edge &b) {
        return a.weight < b.weight;
    });
    int ans = 0;
    int c = 0;
    for (int i = 0; i < edges.size(); i++) {
        int a = edges[i].a;
        int b = edges[i].b;
        int w = edges[i].weight;
        if (!uf.sameSet(a, b)) {
```

```
            ans += w;
            uf.join(a, b);
            c++;
        }
        if (c == n - 1) break;
    }
    return ans;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> g(n + 1, vector<int>(n + 1, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> g[i][j];
        }
    }
    int ans = kruskalsMST(g);
    cout << "Minimum Spanning Weight: ";
    cout << ans << '\n';
    return 0;
}
```

Input

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
```

Output

```
Minimum Spanning Weight: 37
```

8.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;

int fp(vector<int> p, int i) {
    if (p[i] < 0)
        return i;
    return fp(p, p[i]);
}
bool unionbyweig(vector<int> &p, int u, int v) {
    int pu = fp(p, u);
    int pv = fp(p, v);
    if (pu != pv) {
        if (p[pu] <= p[pv]) {
            p[pu] += p[pv];
            p[pv] = pu;
        } else {
            p[pv] += p[pu];
            p[pu] = pv;
        }
        return true;
    }
    return false;
}

int kruskal(vector<vector<int>> &v, int n) {
    int ans = 0;
    vector<pair<int, pair<int, int>>> g;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (v[i][j] != 0)
                g.push_back(make_pair(v[i][j], make_pair(i, j)));
    sort(g.begin(), g.end(), greater<pair<int, pair<int, int>>>());
    vector<int> p(n, -1);
    for (auto i : g) {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (unionbyweig(p, u, v))
            ans = ans + w;
    }
    return ans;
}

int main() {
    int n, t;
    cin >> n;
    vector<vector<int>> v;
    vector<int> vec;
    for (int i = 0; i < n; i++) {
```

```
        vec.clear();
        for (int j = 0; j < n; j++) {
            cin >> t;
            vec.push_back(t);
        }
        v.push_back(vec);
    }
    cout << "Maximum spanning weight : " << kruskal(v, n);
}
```

Input

```
7
0 0 7 5 0 0 0
0 0 8 5 0 0 0
7 8 0 9 7 0 0
5 0 9 0 15 6 0
0 5 7 15 0 8 9
0 0 0 6 8 0 11
0 0 0 0 9 11 0
```

Output

```
Maximum spanning weight : 59
```

9 Week 9

9.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, i, j, k, w;
    cin >> n;
    int graph[n][n];
    string temp;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            cin >> temp;
            if (temp != "INF") {
                graph[i][j] = stoi(temp);
            } else {
                graph[i][j] = 1e8;
            }
        }
    }
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (graph[i][k] + graph[k][j] < graph[i][j]) {
                    graph[i][j] = graph[i][k] + graph[k][j];
                }
            }
        }
    }
    cout << "The shortest path matrix: " << endl;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if(graph[i][j] >= 1e8) cout << "INF";
            else cout << graph[i][j];
            cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input

```
5
0 10 5 5 INF
INF 0 5 5 5
INF INF 0 INF 10
INF INF INF 0 20
INF INF INF 5 0
```

Output

```
The shortest path matrix:
0 10 5 5 15
INF 0 5 5 5
INF INF 0 15 10
INF INF INF 0 20
INF INF INF 5 0
```

9.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<double> items(n);
    vector<double> val(n);
    vector<vector<double>> job;
    for (int i = 0; i < n; i++) {
        cin >> items[i];
    }
    for (int i = 0; i < n; i++) {
        cin >> val[i];
        job.push_back({val[i] / items[i], items[i], (double)(i + 1)});
    }
    double k;
    cin >> k;
    sort(job.rbegin(), job.rend());
    vector<pair<double, double>> ls;
    float profit = 0;
    for (int i = 0; i < n; i++) {
        if (job[i][1] >= k) {
            profit += k * job[i][0];
            ls.push_back(make_pair(k, job[i][2]));
            break;
        } else {
            profit += job[i][1] * job[i][0];
        }
        ls.push_back(make_pair(job[i][1], job[i][2]));
        k = k - job[i][1];
    }
    cout << "Maximum Value : " << profit << endl;
    cout << "Item - Weight" << endl;
    for (auto it : ls)
        cout << it.second << " - " << it.first << endl;
    return 0;
}
```

Input

```
6
6 10 3 5 1 3
6 2 1 8 3 5
16
```

Output

```
Maximum Value : 22.3333
Item - Weight
5 - 1
6 - 3
4 - 5
1 - 6
3 - 1
```

9.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    priority_queue<int, vector<int>, greater<int>> minheap;
    for (int i = 0; i < n; i++) {
        minheap.push(a[i]);
    }
    int ans = 0;
    while (minheap.size() > 1) {
        int e1 = minheap.top();
        minheap.pop();
        int e2 = minheap.top();
        minheap.pop();
        ans += e1 + e2;
        minheap.push(e1 + e2);
    }
    cout << ans;
    return 0;
}
```

Input

```
10
10 5 100 50 20 15 5 20 100 10
6
5 10 15 20 50 100
```

Output

```
895
```

10 Week 10

10.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<vector<int>> v(n, vector<int>(3));
    for (int i = 0; i < n; i++) {
        cin >> v[i][0];
        v[i][2] = i;
    }
    for (int i = 0; i < n; i++) {
        cin >> v[i][1];
    }
    sort(v.begin(), v.end(), [&](vector<int> &a, vector<int> &b) { return a[1]
    < b[1]; });
    int take = 1;
    int end = v[0][1];
    vector<int> ans;
    ans.push_back(v[0][2] + 1);
    for (int i = 1; i < n; i++) {
        if (v[i][0] >= end) {
            take++;
            end = v[i][1];
            ans.push_back(v[i][2] + 1);
        }
    }
    cout << "Non-conflicting activities: " << take << endl
         << "Selected activities: ";
    for (int i : ans)
        cout << i << ' ';
}
```

Input

```
10
1 3 0 5 3 5 8 8 2 12
4 5 6 7 9 9 11 12 14 16
```

Output

```
Non-conflicting activities: 4
Selected activities: 1 4 7 10
```

10.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<vector<int>> v(n, vector<int>(3));
    for (int i = 0; i < n; i++) {
        cin >> v[i][0];
        v[i][2] = i;
    }
    for (int i = 0; i < n; i++) {
        cin >> v[i][1];
    }
    sort(v.begin(), v.end(), [&](vector<int> a, vector<int> b) {
        if (a[1] == b[2])
            return a[0] < b[0];
        return a[1] < b[1];
    });
    priority_queue<pair<int, int>> maxheap;
    int current_time = 0;
    vector<int> selected(n, 0);
    for (int i = 0; i < n; i++) {
        if (current_time + v[i][0] <= v[i][1]) {
            current_time += v[i][0];
            maxheap.push({v[i][0], v[i][2]});
            selected[v[i][2]] = 1;
        } else if (maxheap.size()) {
            if (current_time - maxheap.top().first + v[i][0] <= v[i][1] &&
                maxheap.top().first > v[i][0]) {
                selected[maxheap.top().second] = 0;
                maxheap.pop();
                current_time = current_time - maxheap.top().first + v[i][0];
                selected[v[i][2]] = 1;
                maxheap.push({v[i][0], v[i][2]});
            }
        }
    }
    int total = 0;
    vector<int> ans;
    for (int i = 0; i < n; i++) {
        total += selected[i];
        if (selected[i]) ans.push_back(i + 1);
    }
    cout << "Max number of tasks = " << total << endl
         << "Selected task numbers : ";
    for (int i : ans)
        cout << i << ' ';
}
```

Input

```
7
2 1 3 2 2 2 1
2 3 8 6 2 5 3
```

Output

```
Max number of tasks = 4
Selected task numbers : 2 3 6 7
```

10.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;

int binary_search(vector<int> &a, int key, bool first_occ) {
    int l = a.size();
    int b = 0, e = l - 1;
    while (b <= e) {
        int mid = (b + e) / 2;
        if (a[mid] == key) {
            if (first_occ) {
                if (mid == 0 || a[mid - 1] < a[mid])
                    return mid;
                else
                    e = mid - 1;
            } else {
                if (mid == l - 1 || a[mid + 1] > a[mid])
                    return mid;
                else
                    b = mid + 1;
            }
        } else if (a[mid] > key)
            e = mid - 1;
        else
            b = mid + 1;
    }
    return -1;
}

int main() {
    int n;
    cin >> n;
    vector<int> a;
    for (int i = 0; i < n; ++i) {
        int x;
        cin >> x;
        a.push_back(x);
    }
    sort(a.begin(), a.end());
    int c = 0, i = 0;
    while (i < n) {
        int fo = binary_search(a, a[i], true), lo = binary_search(a, a[i],
   false);
        int nc = lo - fo + 1;
        if (nc > c)
            c = nc;
        i = lo + 1;
    }
    cout << (n * .5 < c ? "yes" : "no") << endl;
    if (n & 1)
```

```
        cout << a[n / 2] << endl;
    else
        cout << (a[n / 2] + a[n / 2 + 1]) / 2.0;
    return 0;
}
```

Input

```
9
4 4 2 3 2 2 3 2 2
```

Output

```
yes
2
```

11 Week 11

11.1 Question 1

Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also find the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include <bits/stdc++.h>
using namespace std;

int dp[1003][1003];
int solve(vector<int> &arr, int i, int j) {
    if (i >= j) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    dp[i][j] = INT_MAX;
    for (int k = i; k < j; k++) {
        dp[i][j] = min(dp[i][j],
                       solve(arr, i, k) + solve(arr, k + 1, j) +
                           arr[i - 1] * arr[k] * arr[j]);
    }
    return dp[i][j];
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> arr[i - 1];
        cin >> arr[i];
    }
    memset(dp, -1, sizeof(dp));
    cout << solve(arr, 1, n) << '\n';
    return 0;
}
```

Input

```
3
10 30
30 5
5 60
```

Output

```
4500
```

11.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find the total number of comparisons for each input case. Time complexity = $O(n * \log(n))$, where n is the size of input).

Program

```
#include <bits/stdc++.h>
using namespace std;

int memo(int i, int current, vector<vector<int>> &dp, vector<int> &coin, int n
    ) {
    if (i == n) {
        return (current == 0);
    }
    if (current < 0)
        return 0;
    if (current == 0)
        return 1;
    if (dp[i][current] != -1)
        return dp[i][current];
    return dp[i][current] = memo(i + 1, current, dp, coin, n) +
                            memo(i, current - coin[i], dp, coin, n);
}
int main() {
    int n;
    cin >> n;
    int target;
    vector<int> coins(n);
    for (int i = 0; i < n; i++) {
        cin >> coins[i];
    }
    cin >> target;
    vector<vector<int>> dp(n, vector<int>(target + 1, -1));
    cout << memo(0, target, dp, coins, n);
}
```

Input

```
4
2 5 6 3
10
```

Output

```
5
```

11.3 Question 3

Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array `arr[n]` , search at the indexes `arr[0], arr[2], arr[4],.....,arr[2^k]` and so on. Once the interval $arr[2^k] < key < arr[2^{k+1}]$ is found, perform a linear search operation from the index 2^k to find the element key. (Complexity $< O(n)$, where n is the number of elements need to be scanned for searching)

Program

```
#include <bits/stdc++.h>
using namespace std;
int memo(int i, int current, vector<vector<int>> &dp, vector<int> arr, int n)
    {
    if (current == 0)
        return 1;
    if (current < 0)
        return 0;
    if (i == n)
        return 0;
    if (dp[i][current] != -1)
        return dp[i][current];
    return dp[i][current] = memo(i + 1, current - arr[i], dp, arr, n) ||
                            memo(i + 1, current, dp, arr, n);
}
int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    int sum = 0;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        sum += arr[i];
    }
    if (sum % 2 == 1) {
        cout << "No" << endl;
    } else {
        vector<vector<int>> dp(n + 1, vector<int>(sum / 2 + 1, -1));
        if (memo(0, sum / 2, dp, arr, n)) {
            cout << "Yes" << endl;
        } else {
            cout << "No" << endl;
        }
    }
}
```

Input

```
7
1 5 4 11 5 14 10
```

Output

```
Yes
```