

Divide and conquer

Change the programming language to C

Merge sort

q2.

```
#include <stdio.h>
```

```
// Merge array a and b into array c
```

```
void merge(int a[], int size_a, int b[], int size_b, int c[]) {
```

```
    int idx1 = 0, idx2 = 0;
```

```
    int idx = 0;
```

```
    while (idx1 < size_a && idx2 < size_b) {
```

```
        if (a[idx1] < b[idx2]) {
```

```
            c[idx] = a[idx1];
```

```
            idx1++;
```

```
        } else {
```

```
            c[idx] = b[idx2];
```

```
            idx2++;
```

```
        }
```

```
        idx++;
```

```
    }
```

```
    while (idx1 < size_a) {
```

```
        c[idx] = a[idx1];
```

```
        idx1++;
```

```
        idx++;
```

```
    }
```

```
    while (idx2 < size_b) {
```

```
        c[idx] = b[idx2];
```

```
        idx2++;
```

```
        idx++;
```

```
    }
```

```
}
```

```
int main() {
```

```
    int n, m;
```

```
    scanf("%d", &n);
```

```
    int a[n];
```

```
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &a[i]);
```

```
    }
```

```
    scanf("%d", &m);
```

```

int b[m];
for (int i = 0; i < m; i++) {
    scanf("%d", &b[i]);
}

int c[n + m];
merge(a, n, b, m, c);

for (int i = 0; i < n + m; i++) {
    printf("%d ", c[i]);
}
printf("\n");

return 0;
}

```

q3.

```
#include <stdio.h>
```

```

void Merge(int a[], int size_a, int b[], int size_b, int c[]) {
    int idx1 = 0, idx2 = 0;
    int idx = 0;

    while (idx1 < size_a && idx2 < size_b) {
        if (a[idx1] < b[idx2]) {
            c[idx] = a[idx1];
            idx1++;
        } else {
            c[idx] = b[idx2];
            idx2++;
        }
        idx++;
    }

    while (idx1 < size_a) {
        c[idx] = a[idx1];
        idx1++;
        idx++;
    }

    while (idx2 < size_b) {
        c[idx] = b[idx2];
        idx2++;
        idx++;
    }
}

```

```
void Sort(int a[], int size_a) {
```

```

    if (size_a == 1) {
        return;
    }

    int mid = size_a / 2;

    int sz1 = mid;
    int sz2 = size_a - mid;

    int a1[sz1];
    int a2[sz2];

    for (int i = 0; i < mid; i++) {
        a1[i] = a[i];
    }

    for (int i = mid; i < size_a; i++) {
        a2[i - mid] = a[i];
    }

    Sort(a1, sz1);
    Sort(a2, sz2);

    Merge(a1, sz1, a2, sz2, a);
}

int main() {
    int n;
    scanf("%d", &n);

    int a[n];
    for(int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    Sort(a, n);

    for(int i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");

    return 0;
}

```

Q5.

```
#include <stdio.h>
```

```
const int mod = 1000000007;
```

```

void merge(int a[], int size_a, int b[], int size_b, int c[]) {
    int idx1 = 0, idx2 = 0;
    int idx = 0;

    while (idx1 < size_a && idx2 < size_b) {
        if (a[idx1] < b[idx2]) {
            c[idx++] = a[idx1++];
        } else {
            c[idx++] = b[idx2++];
        }
    }

    while (idx1 < size_a) {
        c[idx++] = a[idx1++];
    }

    while (idx2 < size_b) {
        c[idx++] = b[idx2++];
    }
}

int countInversions(int a[], int size_a, int b[], int size_b) {
    int ptr = size_b - 1;
    int ans = 0;
    for (int i = 0; i < size_a; i++) {
        while (ptr >= 0 && b[ptr] < a[i]) {
            ptr--;
        }
        ans = (ans + (size_b - 1 - ptr)) % mod;
    }
    return ans;
}

int sortAndCount(int a[], int size_a) {
    if (size_a == 1) {
        return 0;
    }

    int mid = size_a / 2;

    int sz1 = mid;
    int sz2 = size_a - sz1;

    int a1[sz1];
    int a2[sz2];

    for (int i = 0; i < mid; i++) {
        a1[i] = a[i];
    }

```

```

    }

    for (int i = mid; i < size_a; i++) {
        a2[i - mid] = a[i];
    }
    int left = sortAndCount(a1, sz1);
    int right = sortAndCount(a2, sz2);

    // Number of inversions when both halves are put together
    int num = countInversions(a1, sz1, a2, sz2);

    merge(a1, sz1, a2, sz2, a);

    return (((left + right) % mod) + num) % mod;
}

int main() {
    int t;
    scanf("%d", &t);

    while (t--) {
        int n;
        scanf("%d", &n);

        int a[n];
        for (int i = 0; i < n; i++) {
            scanf("%d", &a[i]);
        }

        printf("%d\n", sortAndCount(a, n));
    }

    return 0;
}

```

Quick sort

q2.

#include <stdio.h>

// Replace '_' to solve the problem

```

void partition(int a[], int size_a) {
    int pivot = a[size_a - 1];
    int idx = 0;

    for (int i = 0; i < size_a; i++) {
        if (a[i] <= pivot) {
            int temp = a[idx];

```

```

        a[idx] = a[i];
        a[i] = temp;
        idx++;
    }
}

int temp = a[idx];
a[idx] = a[size_a - 1];
a[size_a - 1] = temp;
}

int main() {
    int n;
    scanf("%d", &n);

    int a[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    partition(a, n);

    for (int i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");

    return 0;
}

```

q3.

Click submit

Q5.

Click submit

Binary search

q1.

```
#include <stdio.h>
```

```
// Function to search for an element in the array
```

```
int Search(int arr[], int x, int n) {
```

```
    for (int i = 0; i < n; i++) {
```

```
        if (arr[i] == x) {
```

```
            return i + 1; // Return 1-based index
```

```
        }
```

```
    }
```

```
    return -1; // Element not found
```

```
}
```

```

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        int n;
        scanf("%d", &n);
        int x;
        scanf("%d", &x);

        int arr[n];
        for (int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
        }
        printf("%d\n", Search(arr, x, n));
    }
    return 0;
}

```

q2.

```
#include <stdio.h>
```

```
// Function to search for an element in the array using binary search
```

```
int binary_search(int arr[], int size_arr, int target) {
```

```
    int left = 0, right = size_arr - 1;
```

```
    while(right >= left) {
```

```
        int mid = (left + right) / 2;
```

```
        if(arr[mid] == target) {
```

```
            return mid + 1;
```

```
        }
```

```
        else if(arr[mid] > target) {
```

```
            right = mid - 1;
```

```
        }
```

```
        else {
```

```
            left = mid + 1;
```

```
        }
```

```
    }
```

```
    return -1;
```

```
}
```

```
int main() {
```

```
    int t;
```

```
    scanf("%d", &t);
```

```
    while(t--) {
```

```
        int n;
```

```
        scanf("%d", &n);
```

```
        int x;
```

```

scanf("%d", &x);
int arr[n];
for(int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}

// Ensure the array is sorted before performing binary search
for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
        if (arr[i] > arr[j]) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}

printf("%d\n", binary_search(arr, n, x));
}
return 0;
}

```

q5.

upper_bound(A,A+n,num) - A

q6.

#include <stdio.h>

```

int main(void) {
    int t;
    scanf("%d", &t);

    while (t--) {
        int n;
        scanf("%d", &n);

        int scores[n];
        for (int i = 0; i < n; i++) {
            scanf("%d", &scores[i]);
        }

        int count[n];
        for (int i = 0; i < n; i++) {
            count[i] = 0;
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (scores[i] >= scores[j]) {
                    count[i]++;
                }
            }
        }
    }
}

```



```
    }  
  }  
}  
  
int boast = 0;  
for (int i = 0; i < n; i++) {  
    if (count[i] > n - count[i]) {  
        boast++;  
    }  
}  
  
printf("%d\n", boast);  
}  
  
return 0;  
}
```

Multiplication of large integers

q2.

click submit

q4.

click submit