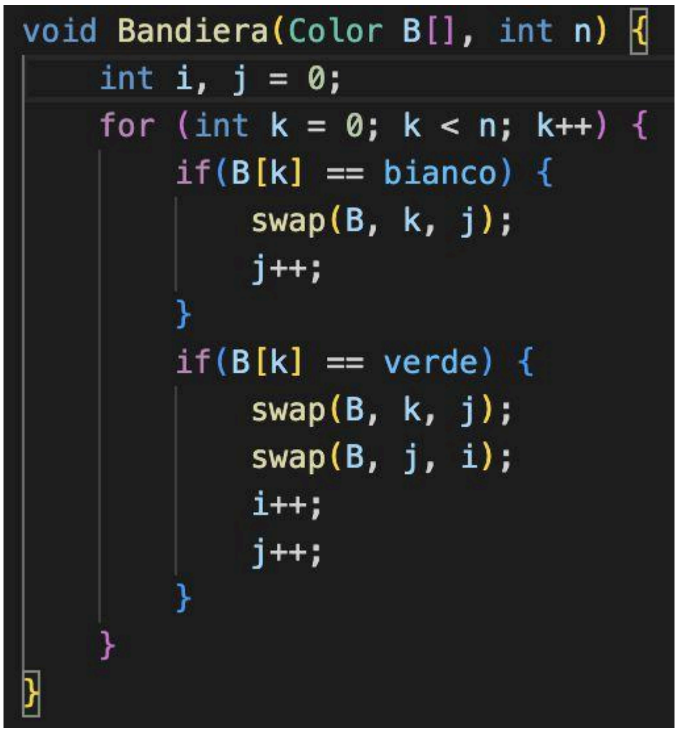
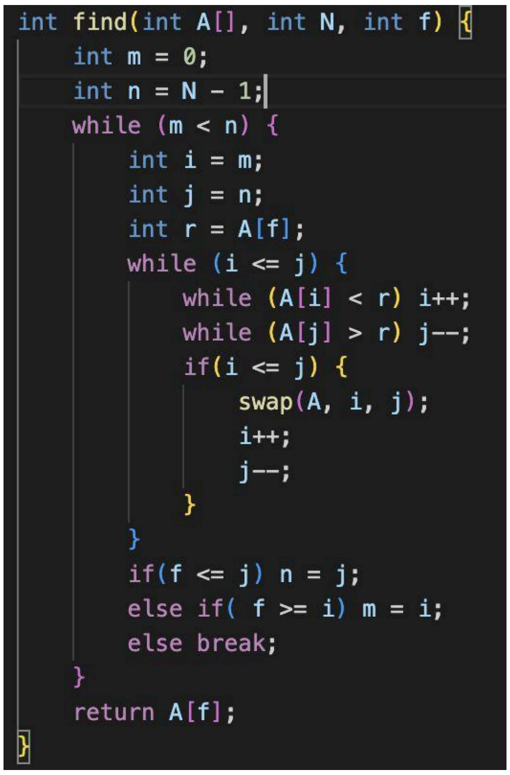
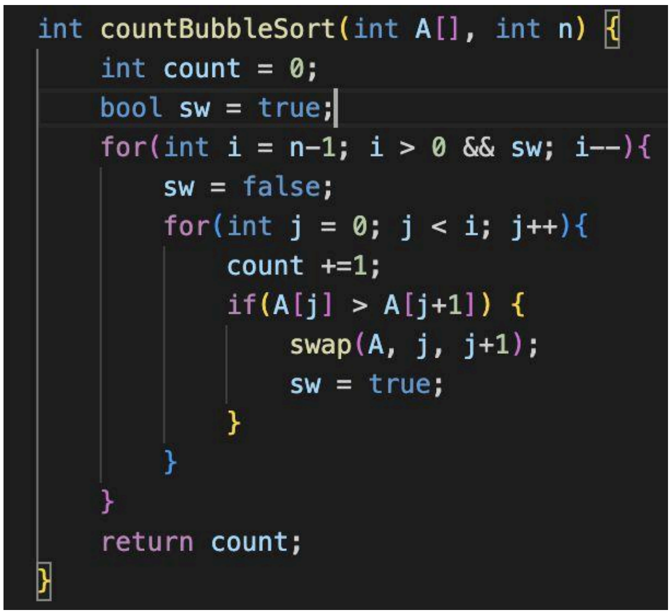


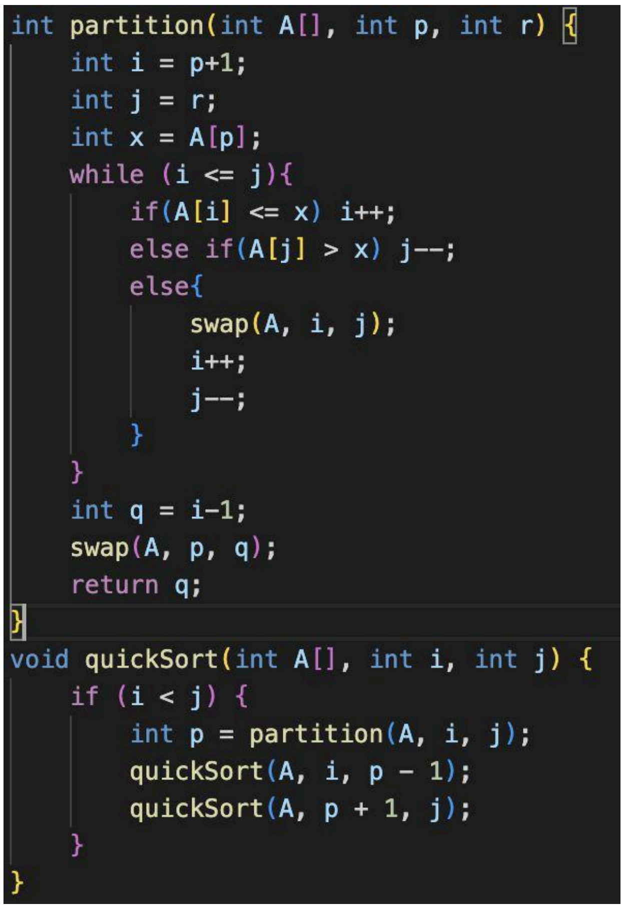
```
void Bandiera(Color B[], int n) {
    int i, j = 0;
    for (int k = 0; k < n; k++) {
        if(B[k] == bianco) {
            swap(B, k, j);
            j++;
        }
        if(B[k] == verde) {
            swap(B, k, j);
            swap(B, j, i);
            i++;
            j++;
        }
    }
}
```



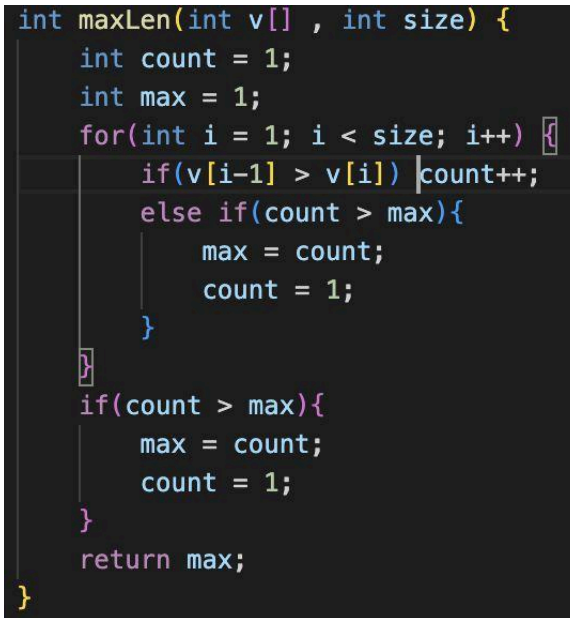
```
int find(int A[], int N, int f) {
    int m = 0;
    int n = N - 1;
    while (m < n) {
        int i = m;
        int j = n;
        int r = A[f];
        while (i <= j) {
            while (A[i] < r) i++;
            while (A[j] > r) j--;
            if(i <= j) {
                swap(A, i, j);
                i++;
                j--;
            }
        }
        if(f <= j) n = j;
        else if( f >= i) m = i;
        else break;
    }
    return A[f];
}
```



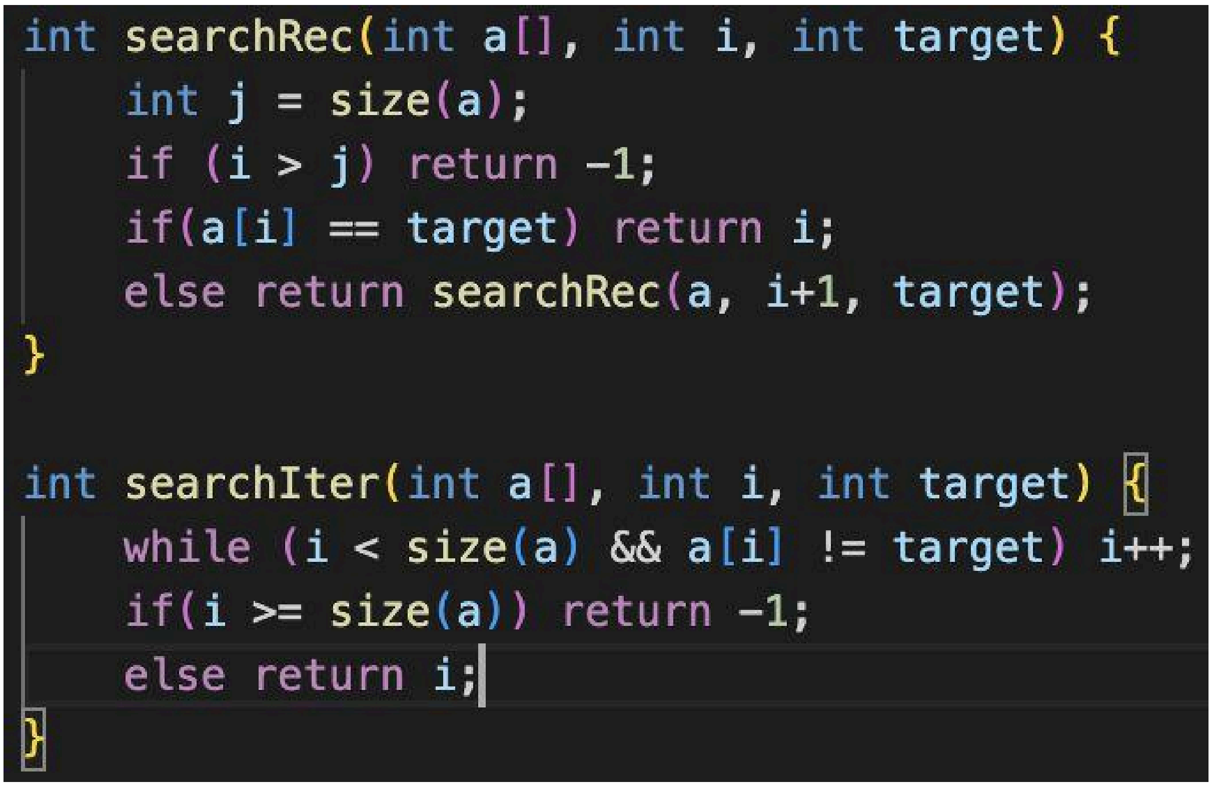
```
int countBubbleSort(int A[], int n) {
    int count = 0;
    bool sw = true;
    for(int i = n-1; i > 0 && sw; i--){
        sw = false;
        for(int j = 0; j < i; j++){
            count +=1;
            if(A[j] > A[j+1]) {
                swap(A, j, j+1);
                sw = true;
            }
        }
    }
    return count;
}
```



```
int partition(int A[], int p, int r) {
    int i = p+1;
    int j = r;
    int x = A[p];
    while (i <= j){
        if(A[i] <= x) i++;
        else if(A[j] > x) j--;
        else{
            swap(A, i, j);
            i++;
            j--;
        }
    }
    int q = i-1;
    swap(A, p, q);
    return q;
}
void quickSort(int A[], int i, int j) {
    if (i < j) {
        int p = partition(A, i, j);
        quickSort(A, i, p - 1);
        quickSort(A, p + 1, j);
    }
}
```

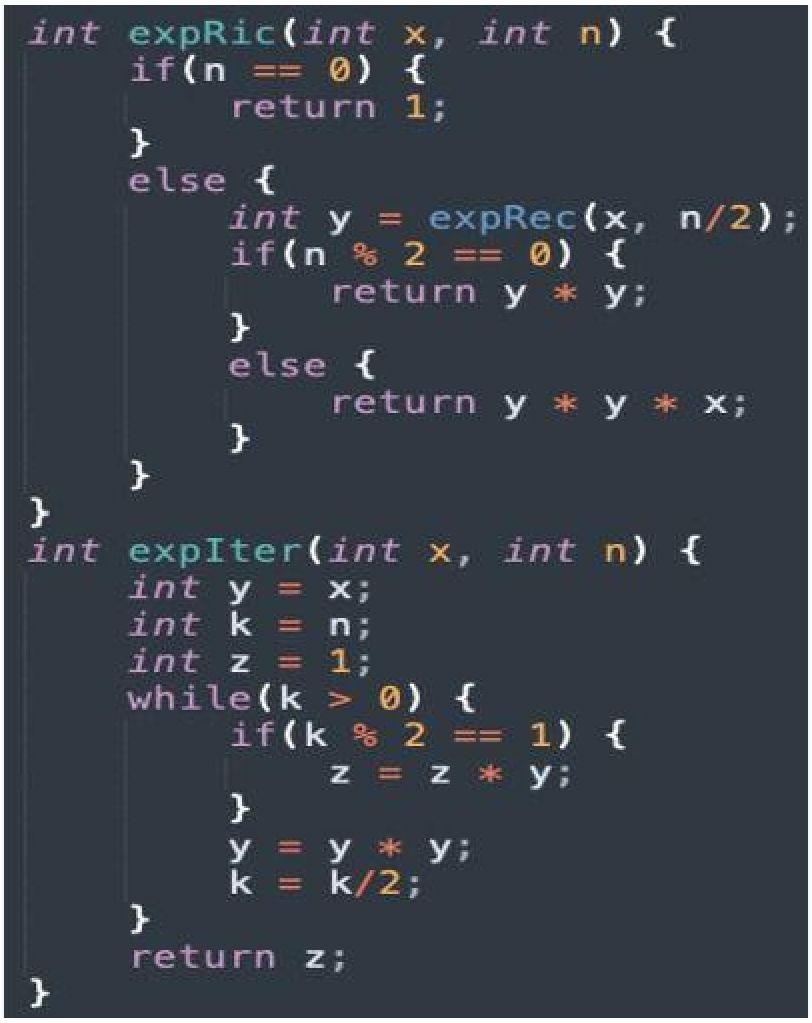


```
int maxLen(int v[] , int size) {
    int count = 1;
    int max = 1;
    for(int i = 1; i < size; i++) {
        if(v[i-1] > v[i]) count++;
        else if(count > max){
            max = count;
            count = 1;
        }
    }
    if(count > max){
        max = count;
        count = 1;
    }
    return max;
}
```

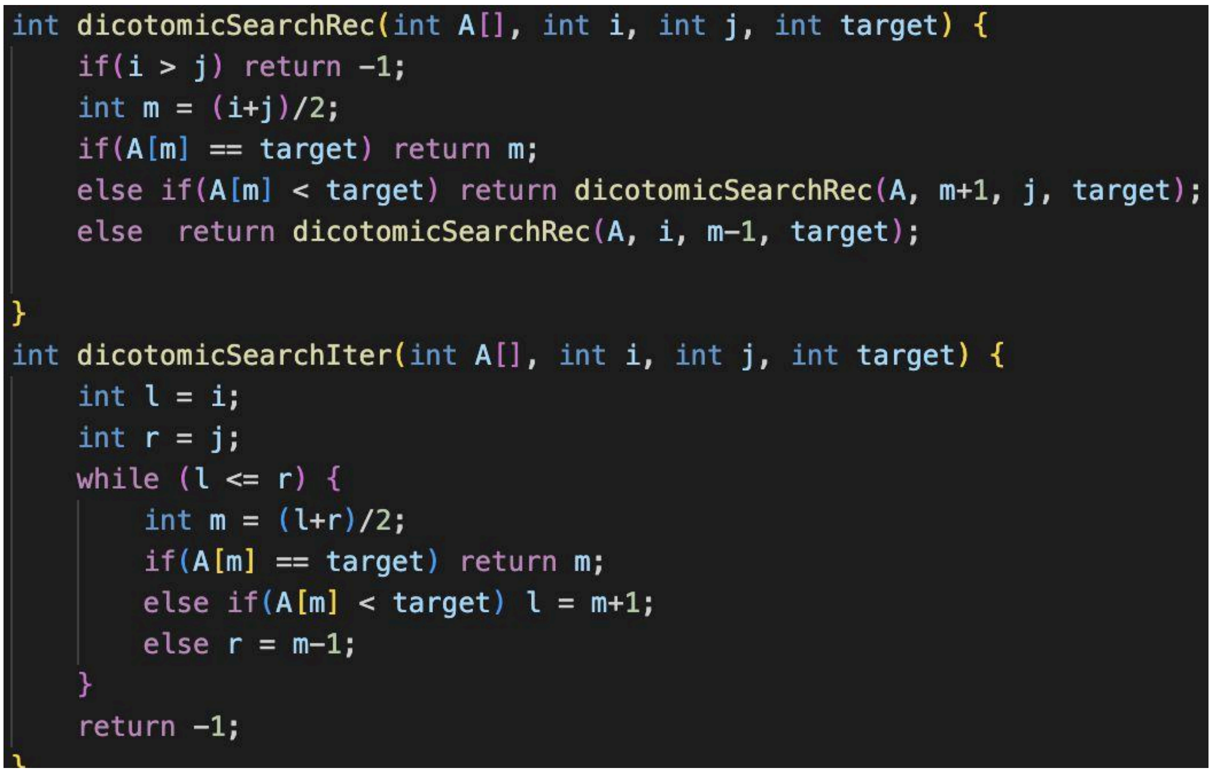


```
int searchRec(int a[], int i, int target) {
    int j = size(a);
    if (i > j) return -1;
    if(a[i] == target) return i;
    else return searchRec(a, i+1, target);
}

int searchIter(int a[], int i, int target) {
    while (i < size(a) && a[i] != target) i++;
    if(i >= size(a)) return -1;
    else return i;
}
```



```
int expRic(int x, int n) {
    if(n == 0) {
        return 1;
    }
    else {
        int y = expRec(x, n/2);
        if(n % 2 == 0) {
            return y * y;
        }
        else {
            return y * y * x;
        }
    }
}
int expIter(int x, int n) {
    int y = x;
    int k = n;
    int z = 1;
    while(k > 0) {
        if(k % 2 == 1) {
            z = z * y;
        }
        y = y * y;
        k = k/2;
    }
    return z;
}
```



```
int dicotomicSearchRec(int A[], int i, int j, int target) {
    if(i > j) return -1;
    int m = (i+j)/2;
    if(A[m] == target) return m;
    else if(A[m] < target) return dicotomicSearchRec(A, m+1, j, target);
    else  return dicotomicSearchRec(A, i, m-1, target);

}
int dicotomicSearchIter(int A[], int i, int j, int target) {
    int l = i;
    int r = j;
    while (l <= r) {
        int m = (l+r)/2;
        if(A[m] == target) return m;
        else if(A[m] < target) l = m+1;
        else r = m-1;
    }
    return -1;
```