4- Sort Colors

Name:	ID:
بدر فيصل عبدالرؤوف عبدالحليم	20210219
زياد محمد حسن عبدالكريم	20210372
خالد محمد سيد مدكور	20210302
زياد محمد خضر البيومي	20210373
زياد امين حسين مسعود	20210359
رشاد سمير رشاد	20210331

Topics

A) The Non-Recursive Solution

B) The Recursive Solution

C) The Non-Recursive Special case Solution

D) Compare between them

1-The Non-Recursive Solution

Pseudocode

```
function bubbleSort(nums):    for pass = 1 to N - 1 do
      for i = 0 to N - pass do          if nums[i] > nums[i+1] then
            swap(nums[i], nums[i+1])    return nums

function main():    nums = array of size N
   for i = 0 to N - 1 do       read nums[i] from user input
   nums = bubbleSort(nums)    for i = 0 to N - 1 do
      print nums[i]     return 0
```

Analysis

Worstcase of Bubble sort: when array elements arranged in descending order and you want to arrange elements in ascending orderin Worstcase the number of passes to sort an array is (n-1)

n is the number of elements in array

first pass: Number of swaps and number of comparisons is [n-1]second pass: Number of swaps and number of comparisons is [n-2]

third pass: Number of swaps and number of comparisons is [n-3]

for last pass: Number of swaps and the number of comparisons = 1

calculate comparisons needed to sort array

= [n-1]+[n-2] + . . . + 1 = n(n-1)/2 times

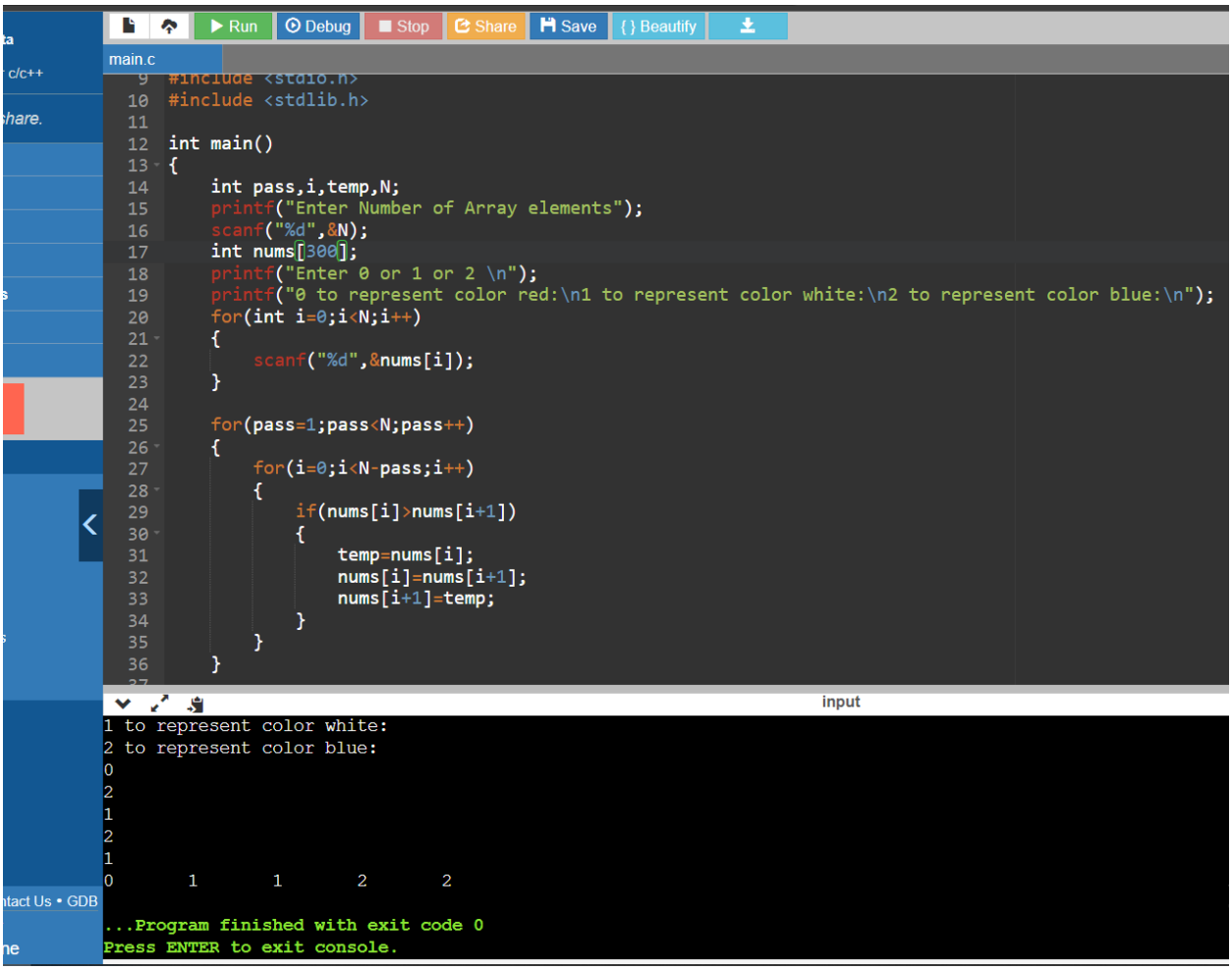
in the Worstcase:number of swaps equal number of comparison

number of comparison (Worstcase) = n(n-1)/2number of swaps (Worstcase) = n(n-1)/2

The worst case when array is in reverse order and it's Time Complexity = O(n^2)

The best case when array is sorted and it's time complexity = O(n)

Screenshot



2-The Recursive Solution

Pseudocode

A) function merge(arr, left, mid, right):

If l < r

1. Find the middle point to divide the array into two halves:

middle m = l + (r-l)/2

2. Copy the data from the input array into the temporary arrays

```
for i from 0 to n1-1:
    L[i] = arr[l + i]
for j from 0 to n2-1:
    R[j] = arr[m + j + 1]
```

3. Compare the elements from the left and right sub-arrays and merge them into a single sorted sub-array

```
i = 0
j = 0
k = left
while i < n1 and j < n2:
    if L[i] <= R[j]:
        arr[k] = L[i]
        i = i + 1
    else:
        arr[k] = R[j]
        j = j + 1
    k = k + 1
```

4. Copy any remaining elements from the left or right sub-arrays into the sorted sub-array in the main function

```
while i < n1:
    arr[k] = L[i]
    i = i + 1
    k = k + 1
  while j < n2:
    arr[k] = R[j]
    j = j + 1
    k = k + 1
```

B) merge_sort(arr[], l, r)

2. Call merge_sort for first half:
   Call merge_sort (arr, l, m)
3. Call merge_sort for second half:
   Call merge_sort (arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
   Call merge(arr, l, m, r)

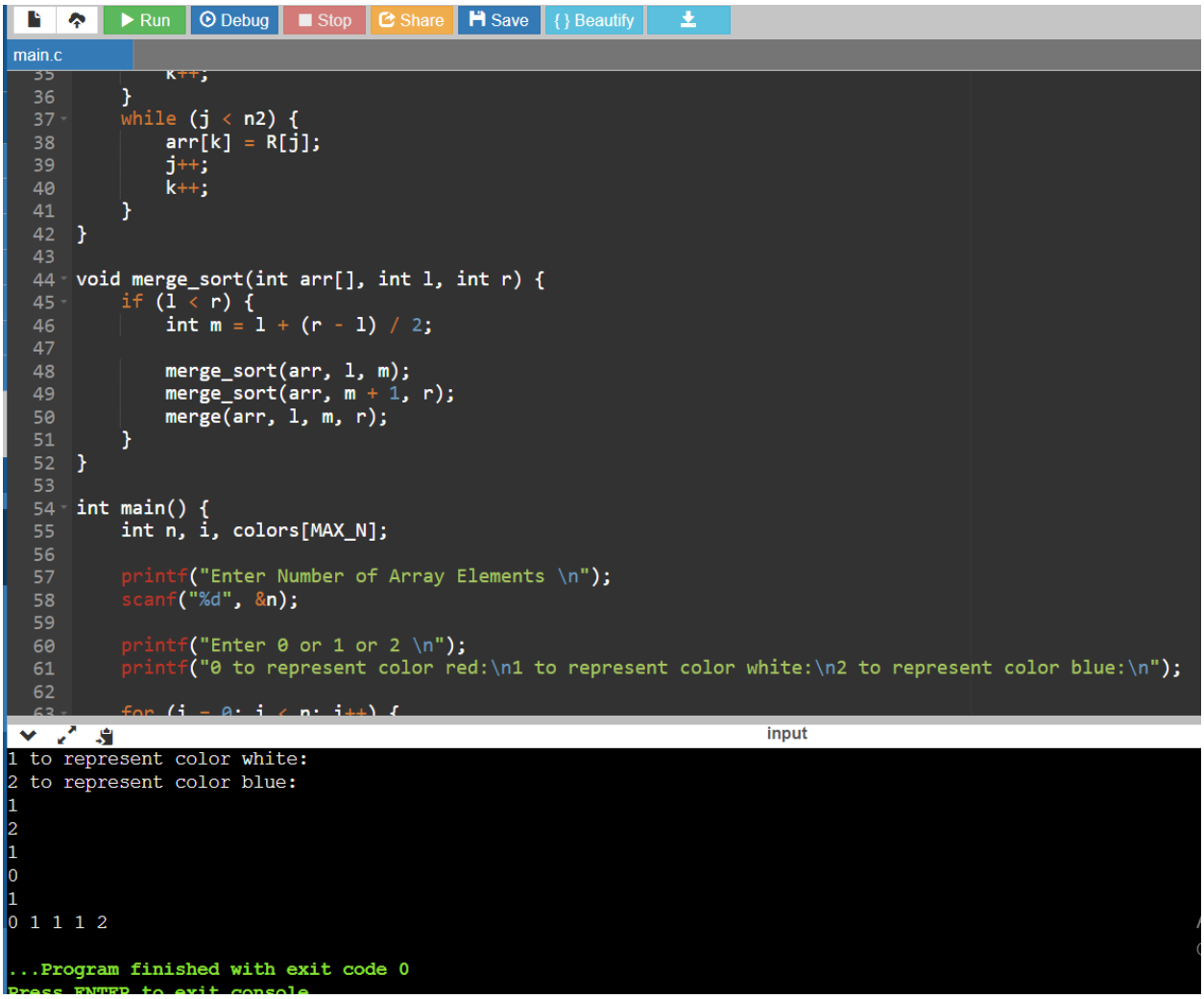
C) function main():
D) scan the input and calling merge_sort(arr[], l, r) and printing the sorted arr[]

Analysis

1- The Merge Sort algorithm has a time complexity of O(n log n) in the worst case, where n is the size of the input array.
2- The merge function takes O(n) time in the worst case to merge two sorted sub-arrays of sizes n/2 each.
3- Therefore, the overall time complexity of the Merge Sort solution is O(n log n).
4- The Merge Sort algorithm has a space complexity of O(n) in the worst case, because it creates temporary arrays of size n/2 during the merge process.
5- Therefore, the overall space complexity of the Merge Sort solution is O(n).

6- The Merge Sort algorithm is a stable sorting algorithm, meaning that it preserves the relative order of equal elements in the input array.
7- In the "Sort Colors" problem, there are only three distinct values, so stability is not a concern.

Screenshot



3-The Non-Recursive Special Case Solution

Pseudocode

FUNCTION swap(a, b):

temp = a

a = b

b = temp

FUNCTION sortColors(nums, numsSize):

i = 0

```
    j = 0
    k = numsSize - 1
    WHILE j <= k:
        IF nums[j] == 0:
            swap(nums[i], nums[j])
            i++
            j++
        ELSE IF nums[j] == 1:
            j++
        ELSE:
            swap(nums[j], nums[k])
            k--
    END WHILE


FUNCTION main():
    nums = [2, 0, 2, 1, 1, 0]
    numsSize = LENGTH(nums)
    sortColors(nums, numsSize)
    FOR i = 0 TO numsSize-1:
        PRINT nums[i]
    END FOR
    RETURN 0
```

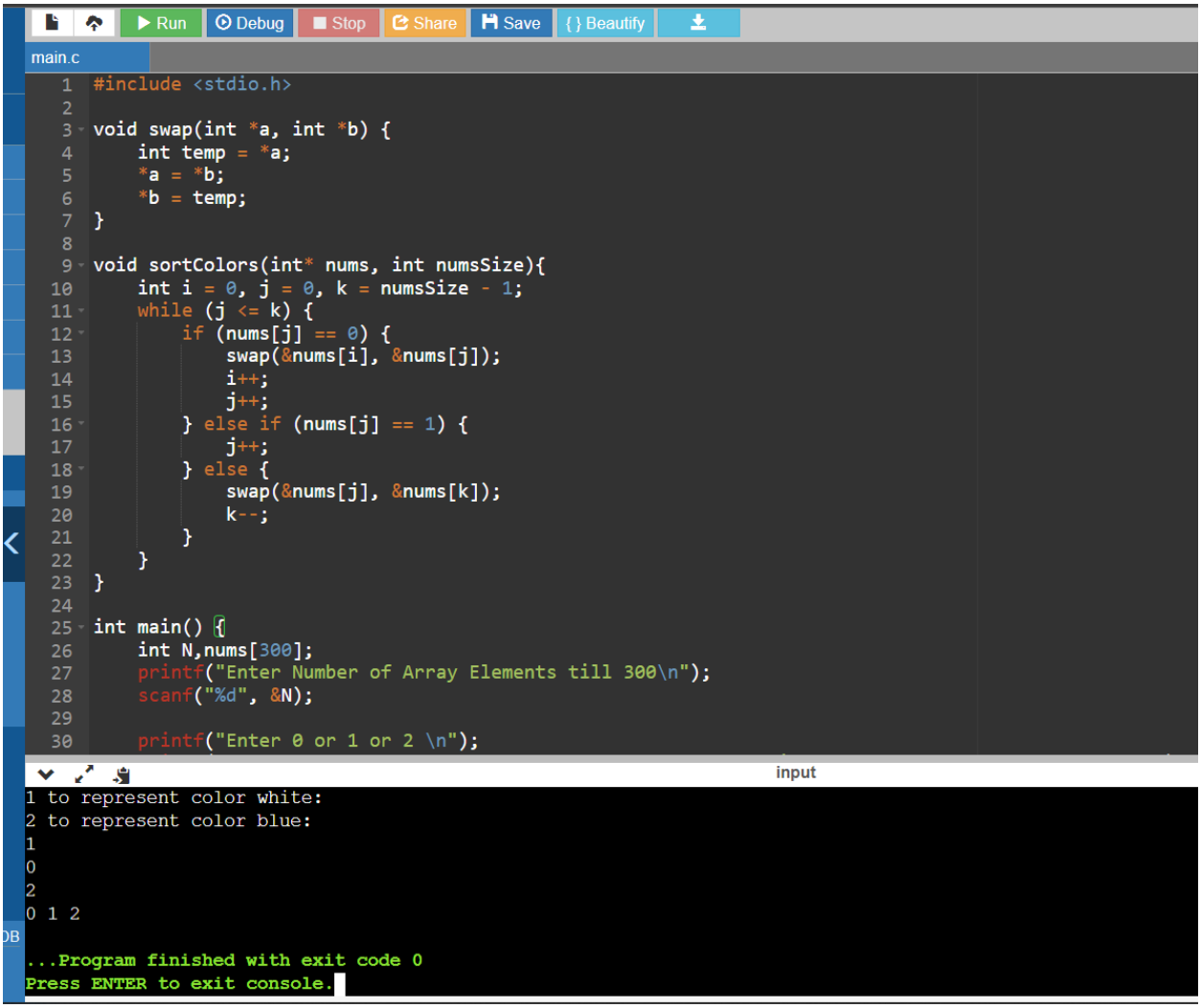
Analysis

The function then uses a while loop to iterate through the array from left to right using the j pointer. If the value at the j index is 0, it swaps the values at the i and j indices, increments i and j to move to the next elements, and continues. If the value at the j index is 1, it simply increments j. If the value at the j index is 2, it swaps the values at the j and k indices, decrements k to move to the next element, but keeps j the same since the value at the new j index needs to be processed further.

The while loop continues until the j pointer meets the k pointer, which indicates that all elements have been processed and the array is now sorted.

The time complexity of the algorithm is O(n) since it processes each element of the array only once. The space complexity is O(1) since it only uses a few pointers to keep track of the sub-arrays.

Screenshot


Run
Debug
Stop
Share
Save
{ } Beautify
main.c
#include <stdio.h>
void swap(int *a, int *b) {
int temp = *a;
*a = *b;
*b = temp;
}
void sortColors(int* nums, int numsSize){
int i = 0, j = 0, k = numsSize - 1;
while (j <= k) {
if (nums[j] == 0) {
swap(&nums[i], &nums[j]);
i++;
j++;
} else if (nums[j] == 1) {
j++;
} else {
swap(&nums[j], &nums[k]);
k--;
}
}
}
int main() {
int N,nums[300];
printf("Enter Number of Array Elements till 300\n");
scanf("%d", &N);
printf("Enter 0 or 1 or 2 \n");
input
1 to represent color white:
2 to represent color blue:
1
0
2
0 1 2
...Program finished with exit code 0
Press ENTER to exit console.

4- The Non-Recursive Special Case Solution

Pseudocode

```
MAX_N = 100
n, i, count_0, count_1, count_2: integer
colors: array of integers with size MAX_N

read n
for i from 0 to n-1 do
   read colors[i]
   switch colors[i] do
      case 0:
         increment count_0 by 1
         break
      case 1:
         increment count_1 by 1
         break
      case 2:
         increment count_2 by 1
         break
   end switch
end for

for i from 0 to n-1 do
   if count_0 > 0 then
      set colors[i] to 0
      decrement count_0 by 1
   else if count_1 > 0 then
```

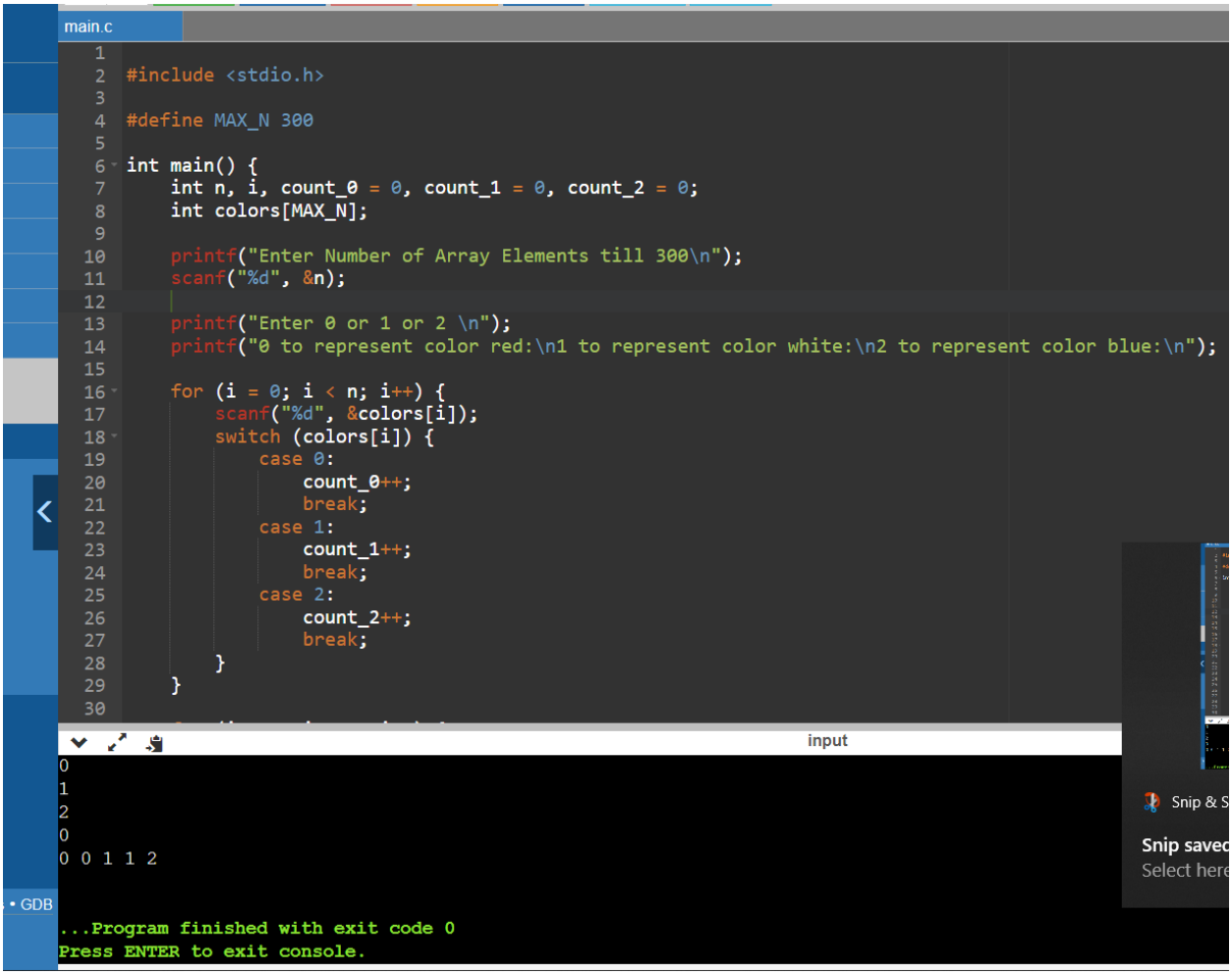
```
        set colors[i] to 1
        decrement count_1 by 1
    else if count_2 > 0 then
        set colors[i] to 2
        decrement count_2 by 1
    end if
end for

for i from 0 to n-1 do
    print colors[i]
end for
```

Analysis

The program sorts the colors array by iterating through it and overwriting each element with the next color in the sequence (0, 1, 2) as long as there are remaining occurrences of that color. This is done using another set of if-else state

The program has a time complexity of O(n), as it only needs to iterate through the colors array twice. The space complexity is O(1), as it only uses a constant amount of additional memory to store the count variables.

Screenshot


main.c
#include <stdio.h>
#define MAX_N 300
int main() {
int n, i, count_0 = 0, count_1 = 0, count_2 = 0;
int colors[MAX_N];
printf("Enter Number of Array Elements till 300\n");
scanf("%d", &n);
printf("Enter 0 or 1 or 2 \n");
printf("0 to represent color red:\n1 to represent color white:\n2 to represent color blue:\n");
for (i = 0; i < n; i++) {
scanf("%d", &colors[i]);
switch (colors[i]) {
case 0:
count_0++;
break;
case 1:
count_1++;
break;
case 2:
count_2++;
break;
}
}
input
0
1
2
0
0 0 1 1 2
...Program finished with exit code 0
Press ENTER to exit console.
GDB
Snip & S
Snip saved
Select here

Compare

Sorting Algorithm	Time Complexity (Best Case)	Time Complexity (Average Case)	Time Complexity (Worst Case)	Space Complexity	Stability	In-Place/Out-of-Place
Merge Sort	Ω(n log n)	Θ(n log n)	O(n log n)	O(n)	Stable	Out-of-Place
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)	Stable	In-Place
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)	Stable	In-Place
sortColors function	Ω(n)	Θ(n)	O(n)	O(1)	Stable	In-Place

--The Best Algorithm between (merge ,Bubble) is Merge sort

--The Best Solution Among all solutions is sortColors Function