CS 137: Assignment #9

Due on Friday, Nov 29, 2024, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2024

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 12.
- <math.h> is not allowed.
- Use Valgrind for testing.

Problem 1

One alternative to the conventional selection sort is the double selection sort algorithm. Double selection sort can be a more efficient sorting algorithm than selection sort due to its ability to sort both the smallest and largest elements in each iteration, and it works as follows:

Find the **first** occurrence of the smallest and largest elements in the array. Swap them with the first and last elements of the array, respectively.

Repeat this process for the remaining elements, considering only the subarray that has not been sorted yet. This means that for each iteration, the range of elements to consider will be from the second element to the second-to-last element of the unsorted subarray.

Create a C program `doubleselect.c` that consists of the function:

```
void doubleselectionsort(int a[], int len);
```

that takes an array of integers and its length, performs double selection sort on the array and prints the array in each iteration. As a result, `len/2` lines should be printed on the screen.

The function to print an array is provided below.

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion). However, **you should keep the required included libraries and printarr function.**

The sample code for testing is below.

```c
#include <stdio.h>

void printarr(int a[], int len){
    for (int i=0; i<len-1; i++)
       printf("%d ", a[i]);
    printf("%d\n", a[len-1]);
}
void doubleselectionsort(int a[], int len);

int main(void) {
    int a[7] = {4, 4, 4, 0, 0, -10, -10};
    doubleselectionsort(a,7);
    int a2[5] = {6, 11, 2, -4, -1};
    doubleselectionsort(a2,5);
    int a3[10] = {1, 8, 5, 4, 6, 2, 5, 6, 2, 9};
    doubleselectionsort(a3,10);

    return 0;
}

```

The expected output:

-10 4 4 0 0 -10 4

-10 -10 4 0 0 4 4

-10 -10 0 0 4 4 4

-4 -1 2 6 11

-4 -1 2 6 11

1 8 5 4 6 2 5 6 2 9

1 2 5 4 6 2 5 6 8 9

1 2 2 4 6 5 5 6 8 9

1 2 2 4 5 5 6 6 8 9

1 2 2 4 5 5 6 6 8 9

Problem 2

Ternary search is similar to binary search, but with the difference that the search space is divided into three parts instead of two in each iteration. This allows ternary search to eliminate two-thirds of the search space with each iteration, which can be more efficient for large arrays. In this question, we want to perform a ternary search to find the last occurrence of a target integer in a non-decreasing sorted array of integers `L`. This means that the search must continue even after locating the `target` value, since the aim is to find the **last occurrence** of it.

To divide the list into three parts, calculate the indices `mid1` and `mid2` as follows:

`mid1 = beginning + (end - beginning) / 3` (integer division)

`mid2 = end - (end - beginning) / 3` (integer division)

where `beginning` and `end` are initially set to `0` and `len - 1`, respectively (where `len` is the length of array `L`).

If the target is found at `mid2`, the search advances towards higher indices by adjusting the search space to `beginning = mid2 + 1`.

If the `target` is not found at `mid2`, but is found at `mid1`, the search advances towards higher indices by adjusting the search space to `beginning = mid1 + 1` and `end = mid2 - 1`.

If the `target` is not found at `mid1` and `mid2`, the search continues in the part of the list where the last occurrence of the `target` element is most likely to exist. This part should be chosen based on the relative values of `target` and values at `mid1` and `mid2` from one of the following bounds:

```
left part: [beginning, mid1 - 1]
middle part: [mid1 + 1, mid2 - 1]
right part: [mid2 + 1, end]
```

Create a C program `ternary.c` that consists of the function:

```
int ternarylastsearch(int L[], int len, int target);
```

That performs the above ternary search algorithm to return the **index** of the **last occurrence** of the `target` in a non-decreasing sorted array of integers `L`, or `-1` if the `target` does not exist in the array. The function should also print a message indicating the two indices being examined each time the search advances, in the format "Examining indices `mid1` and `mid2`".

You are to submit this file containing only your implemented function (that is, you must delete the test cases portion). However, **you should keep the required libraries and structure definition included.**

The sample code for testing is below.

```c
#include <stdio.h>
#include <assert.h>
int ternarylastsearch(int L[], int len, int target);

int main(void) {
     int a[10] = {1, 2, 3, 4, 5, 6, 6, 6, 9, 100};
    assert(7 == ternarylastsearch(a,10,6));
    printf("\n");
    int a2[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
    assert(-1 == ternarylastsearch(a2,10,100));
    printf("\n");
    assert(-1 == ternarylastsearch(a2,10,11));
    printf("\n");
    assert(0 == ternarylastsearch(a2,10,2));
    printf("\n");
    int a3[12] = {6,6,6,6,6,6,6,6,6,6,6,6};
    assert(11 == ternarylastsearch(a3,12,6));

    return 0;
}

```

The expected output:

Examining indices 3 and 6
Examining indices 7 and 9
Examining indices 8 and 8

Examining indices 3 and 6
Examining indices 7 and 9

Examining indices 3 and 6
Examining indices 4 and 5

Examining indices 3 and 6
Examining indices 0 and 2
Examining indices 1 and 1

Examining indices 3 and 8
Examining indices 9 and 11