Week 8 Assignment

MATH 5061: Fundamentals of Computer Programming for Scientists and Engineers

Due: November 7th, 2016

Instructions

Please submit your programs for this as .cpp file and include the compiled binary program. Send them as a .tar.gz or ZIP file via email to math5061@temple.edu. Make sure to use the subject line (without quotes) "MATH5061:Assignment 08 :ACCESSID" Where ACCESSID is your AccessNet ID, for example tue86537.

1 Text to number conversion

Write a function `string_to_int(char *)` which takes any 0-terminated character string containing a number, such as `"1234"` or `"-1234"`, and returns the corresponding integer number.

> **Hint:** You can check whether a single character is a digit by making sure it is in the range between the characters `'0'` and `'9'`.

> **Hint:** Take advantage of the fact that characters are just small integers and you can do math with them. E.g., `'0'-'0'= 0`, `'1'-'0'= 1`, `'2'-'0'= 2`

Listing 1 shows a small test program which you can use to verify if your function works correctly.

Listing 1: Testing program for your `string_to_int` function

```c
#include <stdio.h>

int string_to_int(char * str) {
    int val;
    // TODO
    return val;
}

int main() {
    char * exampleA = "1234";
    char * exampleB = "100000";
    char * exampleC = "-2000";

    int a = string_to_int(exampleA);
    int b = string_to_int(exampleB);
    int c = string_to_int(exampleC);

    if (a == 1234 && b == 100000 && c == -2000) {
        printf("your function works!\n");
    } else {
        printf("your function doesn't work!\n");
    }

    return 0;
}
```

2 Insertion Sort

In this exercise you will implement the insertion sort algorithm. Here is a basic skeleton for this program.

```
int * insertion_sorted(int * a, int length); // TODO
void  print_array(int * a, int length);      // TODO

int main() {
    int a[] = {10, 7, 2, 5, 1, 4, 9, 3, 6, 8};

    print_array(a, 10);

    int * sorted = insertion_sorted(a, 10);

    print_array(sorted, 10);

    // TODO free dynamic memory of sorted array

    return 0;
}
```

2.1 `print_array` function

The `print_array` function should take a pointer to an array and its length as arguments and print out all of the elements on a single line, separated by commas and surrounded by brackets. Note that there are no additional spaces or commas after the last element.

Listing 2: Output of `print_array` for initial `a` array

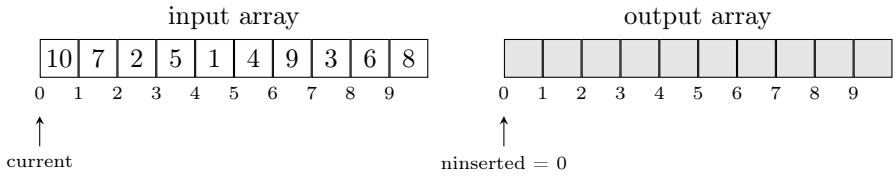
```
[10, 7, 2, 5, 1, 4, 9, 3, 6, 8]
```

2.2 `insertion_sorted` function

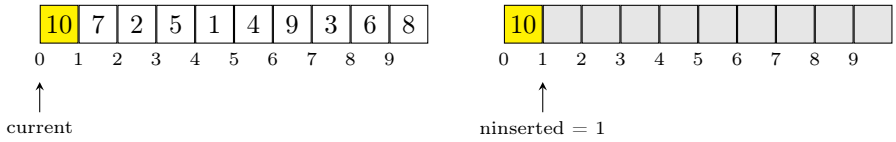
The insertion sort algorithm should be implemented inside of the function `insertion_sorted(int * a, int length)`. In our implementation we do not modify the original array. Instead, we dynamically allocate memory on the heap, copy values to this new array and sort them in ascending order. Finally the function returns the pointer to this new sorted array.

Algorithm

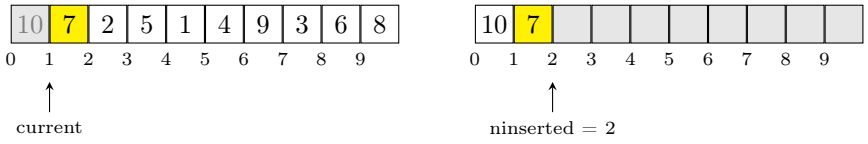
You start out with an unordered input sequence and create an empty output sequence of the same size. We're going to fill up the output array from left to right and keep a counter `ninserted` to store the number of inserted values, which is also the index of the next insertion.



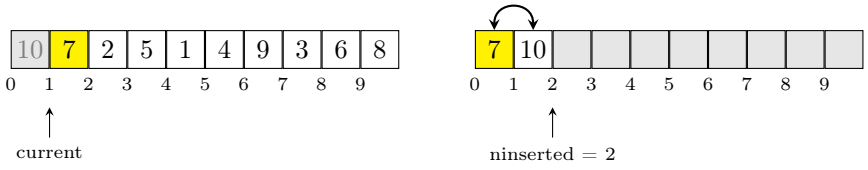
You then take the first element from the input sequence and append it to the output sequence. Since our output sequence is empty, we insert at location `ninserted`, which is 0. Our output sequence has increased by one element, so we increase `ninserted` by 1. This is now the only element in the sequence and it is already in correct order, which means nothing else is to do.



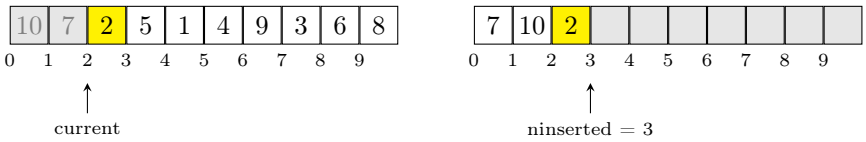
We continue by looking at the next element from the input sequence and copying it to the end of the output sequence at `ninserted = 1`. We now have two elements in the output sequence and therefore increase `ninserted` to 2.



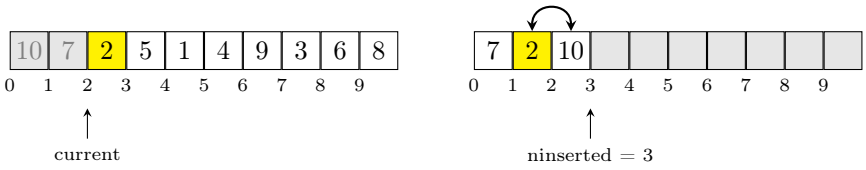
However, the new element is not at the correct location yet. If we compare it with the previous element, we notice that it is larger. So we swap them.



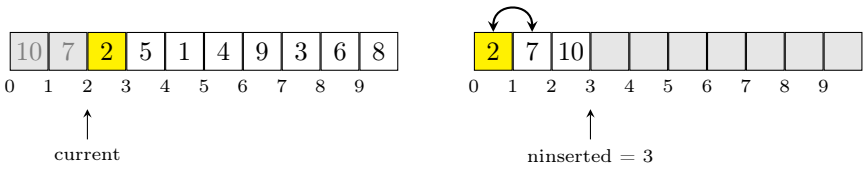
After swapping the sequence is in order. And we can continue with the next element from the input sequence.



We again append the next value from the input sequence to the output sequence and increment `ninserted` to 3. The new element is not at the correct position, so we swap it once.



We continue swapping and moving the new value forward until it reaches the correct location (either the left neighbor is smaller or we reached the beginning of the output sequence).



Follow the same steps until all elements from the input sequence have been copied and moved to the correct position in the output sequence.

Summary: The insertion sort algorithm goes through the original sequence one by one and copies the values to a new array. While doing this, it will check if the newly inserted value is at the correct position. If not it will move it to the correct location by swapping.

Hint #1: You will need a nested loop. An outer loop, which copies values from input to output array, and an inner loop which will move the new element in the output array to the correct location. Use variables to store indices of all the elements you are looking at during an iteration.

Hint #2: Use the `print_array` function inside your sorting loop to help you verify the individual steps and check that your code is doing what you think it is doing. Once you got it working, remove these statements.