Assignment 6 (V1.1)

Design: Sorting

Linhao Chen
CSE 13S - Fall 2019

Date:11/05/2019

1 Introduction

This program will try to run a program for sorting c-size random array in order by using different algorithms. By selecting specific sorting method, the program can print first n elements of c-size array which created by using random seed r. The default value for n and c will be 100, while random seed will be 8222022. Expect to use inttypes, getopt, loop, dynamic memory allocation, etc.

2 Sorting

Main {
Set three variables for random_seed; array_length; print_length
Set the default value of variables above to 8222022, 100, 100
Set four bool variables to represent whether we need to use this sort method
Set the default of bool variables to false
Read the command line
Print error and exit the program if arguments does not satisfy the requirement
Switch to case depend on the command-line option
In case ‘n’
Set the value of array_length to parameter after -n

In case ‘r’
Set the value of random_seed to parameter after -r

In case ‘p’
Set the value of print_length to parameter after -p

In case ‘a’
Set variable a_check to true

In case ‘b’
Set variable b_check to true

In case ‘c’
Set variable c_check to true

In case ‘d’
Set variable d_check to true

In case ‘A’
Set all check Boolean variables to true

If (a_check is true)
Create pseudo-random array by calling array_create by using parameter array_length and random_seed)
Call Function sortA to sort the array
Decide how many numbers need to print (smaller element between array_length and print_length)
Print array by calling function array_print
Delete the array for deallocating the memory

If (b_check is true)
Create pseudo-random array by calling array_create by using parameter array_length and random_seed)
Call Function sortB to sort the array
Decide how many numbers need to print (smaller element between array_length and print_length)
Print array by calling function array_print
Delete the array for deallocating the memory

If (c_check is true)
Create pseudo-random array by calling array_create by using parameter array_length and random_seed)
Call Function sortC to sort the array
Decide how many numbers need to print (smaller element between array_length and print_length)
Print array by calling function array_print
Delete the array for deallocating the memory

If (d_check is true)
Create pseudo-random array by calling array_create by using parameter array_length and random_seed)
Call Function sortD to sort the array
Decide how many numbers need to print (smaller element between array_length and print_length)
Print array by calling function array_print
Delete the array for deallocating the memory

```
    Program Ends
}

Array_create (parameters random_seed and array_length) {
    Initialize the pseudo random by using seed
    Allocate memory space for array
    For (0 to array_length)
        Generate a new random number set put it to array
        Use bit mask to set the random number
    Return array
}

Array_delete (parameter the pointer of array) {
    Free the memory space
    Return
}

Array_print (parameters print_length and array) {
    For (i from 0 to length)
        If (i%7 equals 0)
            Print new line
        Print next elements by using format
    Return
}
```

3 Sort A (Updated)

Most part of codes come from assignment pdf created by professor Long

Use global variables to count compare operation and move operation

```
SortA (parameters array and array_length) {
    For (i from 0 to length-1) {
        Find the index of No.i smallest element in array by calling function
        If (index is not equals to a[i] itself)
            Swap the elements stores in a[i] and a[index]
            Move number plus ~~one~~ three (swap actually move three times, move x to temp, move
                y to x, move temp to y)
    Print format for sorting method (print length, # of move, # of compare operation)
    Return to main
}

Min_index (parameter array, index of i, length of array) {
    For (i to length of array)
        Compare a[i] with a[i+1]
        Compare number plus one
```

```
        Store the index of smaller element
    Return the index of smallest element
}
```

4 Sort B (Updated)

This part is written by using pseudo-code which provided by professor Long

Use global variables to count compare operation and move operation

```
SortB (parameters array and array_length) {
    Set Boolean swapped to true
    For (i from length-1 to zero; do if swapped is true and i bigger than zero; i minus one) {
        Set swapped variable to false;
        For (o from 0 to i-1)
            Compare number plus one
            if(a[o] bigger than a[o+1])
                Swap the elements stores in a[o] and a[o+1]
                Move number plus three (swap actually move three times, move x to temp,
                    move y to x, move temp to y)
                Set swapped vairbale to true
    Print format for sorting method (print length, # of move, # of compare operation)
    Return to main
}
```

5 Sort C(Updated)

This part is written by using pseudo-code which provided by professor Long

Use global variables to count compare operation and move operation

```
SortC (parameters array and array_length) {
    For (i from 1 to length)
        Create a temp element equals to array[i]
        Set j equals to i - 1
        While (j larger than 0 and a[j] bigger than temp element)
            Set a[j+1] equals to a[j]
            Compare number plus one
            Move number plus one
            Set j minus 1
        Compare number plus one when j bigger than 0 (if while loop break, it may cause j < 0 or
            a[j] smaller than temp element, we only need to increase compare counter when a[j] is
            not bigger than temp)
        Set array [j+1] equals to temp element
        Move number plus one
    Print format for sorting method (print length, # of move, # of compare operation)
    Return to main
}
```

6 Sort D(Updated)

This part is written by using pseudo-code which provided by professor Long

Use global variables to count compare operation and move operation

```
SortD (parameters array and array_length) {
   Set gaps by using array {701, 301, 132, 57, 23, 10, 4, 1}
   Set three variables i, j, o
   For (o from 0 to 8)
        gap equals to gaps[o]
        for (i from gap to array_length)
             create a temp element equals to array[i]
             ~~compare number add one~~ (See part below the for loop)
             for (j equals to i; do if (j not smaller than gap and a [j - gap] larger than temp element);
                j minus gap) {
                a [j] equals to a [j - gap]
                compare number add one (compare should increase 1 when having comparison)
                move number add one
             Compare number plus one when j not smaller than gap (if for loop breaks, it may
                cause j < gap or a [j - gap] not bigger than temp element, we only need to increase
                compare counter when a [j - gap] is not bigger than temp)
             Set a [j] equals to temp
             Move number add one
   Print format for sorting method (print length, # of move, # of compare operation)
   Return to main
}
```