Din Bostandzic
CSE 13S

Assignment 5 - Sorting Program Design

Pre-Lab Answers

Part 1

1. 4 rounds of swapping will be required to sort the numbers 8,22,7,9,31,5,13 using bubble sort in ascending order, with a total of 10 comparisons being made.
2. There will be n(n-1)/2 comparisons made in the worst case scenario of bubble sort.

Part 2

1. The worst case time complexity of shell sort depends on the gap size because the bigger the gap, the more passes through the elements are required, thus resulting in more comparisons and swaps taking place. To improve the time complexity of this sort by changing the gap size, it is achieved by decreasing the gap by adjusting the equation for gaps, which will result in less inputs to be checked.
2. To improve the runtime of this sort without changing the gap size, use pre-existing gap values that will optimize the sort or include a binary search to quickly locate where to place the elements.

Part 3

1. The worst case time complexity of O(n^2) for Quicksort only occurs when a pivot point of the starting index or very last index of an array is chosen, which results in having to go through each value. Quicksort is not doomed by this issue because usually the pivot is chosen to be the middle index in the array to get optimized.

Part 4

1. When the binary search algorithm is combined with insertion sort algorithm, the effect is a reduced time complexity, as the items can get their locations of placement more quickly.

Part 5

1. Since each sort will reside in its own header file, to keep track of the number of moves and comparisons, an extern variable for swaps and compares will be used in the main and each sort header where the counts can be accessed as needed.

The program is implementation of binary insertion sort, bubble sort, shell sort, and quick sort. The user gets options of each and can also specify a desired array size, print size of elements, and random seed to create a dynamic array of values with at least 30 bits but no larger than 2^30-1. These options are not mutually exclusive, so can be used in any combination, as explained in README.md. Once the user enters in desired options, the array is created and thus the sort is performed on it. Once sorted a tabular report of the sorted elements is output, including a header of what sort has been performed, and statistics about it including the number of swaps and comparisons that were done in it, as well as the total number of elements in it, which is kept track using extern variables.

The following functional decomposition has been implemented, along with the supporting pseudocode:

```
    3.0 Sorting
        3.1 setopt(in option as pointer to enum opts type, in enum_opts as enum opts)
        3.2 checkopt(in option as pointer to enum opts type, in enum_opts as enum opts)
        3.3 choices(in default_arraysize as integer, in default_seed as integer, in option as
                pointer to enum opts, in default_print as integer)
        3.4 array_maker(in default_size as integer, in default_seed as integer)
        3.5 print_array(in arr as integer pointer, int default_size as integer, in default_print as integer)
        3.6 i_sort_operate(in default_arraysize as integer , in default_seed as integer, in
                 default_print as integer)
        3.7 q_sort_operate(in default_arraysize as integer , in default_seed as integer, in
                     default_print as integer)
        3.8 s_sort_operate(in default_arraysize as integer , in default_seed as integer, in
                     default_print as integer)
        3.9 b_sort_operate(in default_arraysize as integer , in default_seed as integer, in
                     default_print as integer)
```

<u>main module data design</u>

```
Define OPTIONS as constant string "Absqip:r:n:"
Define opts as enumerated type {A, b, s, q, i}
Declare enum opts option initialized to 0
Declare default_arraysize  as integer initialize to 100
Declare default_print as integer initialized to 100
Declare default_seed as integer initialize to 8222022
Define max_num as integer with value (1 << 30)-1
Declare c as character
Declare input_num as string initialized to NULL
```

<u>main module design</u>

```
Begin main(in arc as integer, in argv as string)
    Begin While ((c = getopt(argc, argv,OPTIONS)) != -1)
        Switch(c)
            Case 'A'
                setopt(&option, 0
                Break
            Case 'b'
                setopt(&option, 1)
                Break
            Case ‘s’
                setopt(&option, 2)
                Break
            Case ‘q’
                setopt(&option, 3)
                Break
            Case ‘i’
                setopt(&option, 4)
                Break
            Case ‘p’
                setopt(&option, 5)
                Assign to input_num value of optarg
                Assign to default_print value of input_num as integer
                Break
            Case ‘r’
                Assign to input_num value of optarg
                Assign to default_seed value of input_num as integer
                Break
            Case ‘n’
                Assign to input_num value of optarg
                Assign to default_arraysize value of input_num as integer
                Break
            Default case
                Display “Character not found in string”
                Return exit status fail
        End Switch
    End while
    Begin if (argc == 1)
        Display “No arguments supplied!”
        Return exit status fail
    End if
    Call choices(in default_arraysize as integer, in default_seed as integer, in option as
               address of enum opts type, in default_print as integer)
End main
```

choices module design

```
Begin choices
    Begin for (enum opts x = A, x <= i, x++)
        Begin if (call checkopt(in option as pointer to enum opts, in x as enum opts)
            Begin switch (x)
                case A:
                 Call i_sort_operate(in default_arraysize as integer, in default_seed
                           as integer, in default_print as integer)
                  swaps = 0
                  compares = 0
                Call q_sort_operate( in default_arraysize as integer, in
                        default_seed as integer, in default_print as integer)
                swaps = 0
                compares = 0
                Call s_sort_operate(in default_arraysize as integer, in default_seed
                          as integer, in default_print as integer)
                swaps = 0
                compares = 0
                Call b_sort_operate(in default_arraysize as integer, in default_seed
                              as integer, in default_print as integer)

                break
                case b:
                Call b_sort_operate(in default_arraysize as integer, in default_seed
                              as integer, in default_print as integer)
                break
                case s:
                Call s_sort_operate(in default_arraysize as integer, in default_seed
                          as integer, in default_print as integer)
                break
                case q:
                Call q_sort_operate(in default_arraysize as integer, in default_seed
                              as integer, in default_print as integer)
                break
                case i:
                Call i_sort_operate(in default_arraysize as integer, in default_seed
                          as integer, in default_print as integer)
                break
                default:
                Break
            End switch
        End if
    End for
End choices
```

print_array module design

```
    Begin print_array
        Display “elements, moves, compares” default_size, swaps, compares
        Declare i as integer initialized to 0
        Begin while(i < default_print)
            Begin for(integer j = 1, j < 8, increment j)
                Display arr[i]
                Increment i
                Begin if (i == default_print)
                    Break
                End if
            End for
        End while
    End print_array
```

array_maker module design

```
Begin array_maker
    Call srand(pass in default_seed as integer)
    Create Dynamically allocated array of size default_size initialized all to 0
    Begin for (integer i = 0, i < default_size, increment i)
        Begin if (arr != NULL)
            arr[i] = rand() % max_num
        End if
    End if
    Return arr
End array_maker
```

i_sort_operate module design

```
Begin i_sort_operate
    Display “Binary Insertion Sort”
    Declare arr as pointer to unsigned integer 32 bit type assign value by calling
            array_maker(in default_arraysize as integer, in default_seed as integer)
    Call binary_insertion_sort(in arr as uint32_t, in default_arraysize as integer)
    Call print_array(in arr as uint32_t, in default_arraysize as integer,  in default_print as integer);
    Call free(in arr as uint32_t)
End i_sort_operate
```

q_sort_operate module design

```
Begin q_sort_operate
    Display “Quick Sort”
    Declare arr as pointer to unsigned integer 32 bit type assign value by calling
            array_maker(in default_arraysize as integer, in default_seed as integer)
    Call quick_sort(in arr as uint32_t, 0, in default_arraysize as integer-1)
    Call print_array(in arr as uint32_t, in default_arraysize as integer,  in default_print as integer);
    Call free(in arr as uint32_t)
End q_sort_operate
```

s_sort_operate module design

```
Begin s_sort_operate
    Display “Shell Sort”
    Declare arr as pointer to unsigned integer 32 bit type assign value by calling
            array_maker(in default_arraysize as integer, in default_seed as integer)
    Declare arr2 as pointer to integer assign value by call gap(in default_arraysize as integer)
    Call shell_sort(in arr as uint32_t, in default_arraysize as integer, arr2 as integer)
    Call print_array(in arr as uint32_t, in default_arraysize as integer,  in default_print as integer);
    Call free(in arr as uint32_t)
End s_sort_operate
```

b_sort_operate module design

```
Begin b_sort_operate
    Display “Bubble Sort”
    Declare arr as pointer to unsigned integer 32 bit type assign value by calling
            array_maker(in default_arraysize as integer, in default_seed as integer)
    Call bubble_sort(in arr as uint32_t, in default_arraysize as integer)
    Call print_array(in arr as uint32_t, in default_arraysize as integer,  in default_print as integer);
    Call free(in arr as uint32_t)
End b_sort_operate
```