ASSIGNMENT 4 DESIGN DOCUMENT

Pehara Vidangamachchi

October 23rd 2022

Ethan L. Miller

About:

This assignment aims to create a program that uses several different sorting methods such as Shell Sort, Bubble Sort, and Heapsort based on a provided python pseudo-code. Additionally, we are tasked with creating a test harness that uses set.h to handle the command line options. This assignment will require us to work on the python code that is acting as pseudo-code for our c program. As an end result, the user should be able to use the program in all the listed ways above with a given command.

Design Process for bubble.c

Bubble sort is the most simple algorithm out of the bunch, the way that it works is that iterates through all the elements swapping them if they are not within the right order (lowest to highest). It repeats this process until the program no longer needs to swap anything around anymore. I was able to write my code based on this algorithm by utilizing the pseudo code given to us in python and using multiple for and if statements.

#including any necessary header files

Declaring the bubble sort function:

Initializing the elements that will be used within the code

Creating a for loop for the main bubble sort program:

Initialize swapped as false

Create a for loop for the main math of the sorting algorithm:

Setup the compare stats to keep count of

Create an if statement to sort out the arrays:

Setup the move sats to keep track of

Create a temp array to avoid changing the values later on

Create the main mathematical methods used to sort

Set swapped equal to true

Set it so that when the above is not equal to swapped the program breaks, this will return the results

Design Process for heap.c

The sorting algorithm for heap is clearly given in the example pseudo-code within python. Heap sort is a comparison-based sorting algorithm that iterates through several different values in order to generate an organized list of values. I was able to write my code following the example and using while, if, else and for statements accompanied by the usage of the unsigned 32-bit integer which I heavily utilized throughout each sorting algorithm.

Main Code:

Design Process for quick.c

The algorithm quick sort, also known as the partition-exchange sort, is a sorting algorithm that follows a divide-and-conquer principle. The way it works is by spitting larger

arrays into small arrays and running shell sort if the value is below eight. I was able to implement this code following the given pseudo code and translating it into a way that worked for c. I did this with the use of multiple functions and several for and if statements. Some of my code was similar to that of my bubble sort algorithm so I was able to borrow bits and pieces from it.

#include any necessary header files

#define any macros that will be used

Find pivot algorithm:

Initialize any elements that will be used within the code segment

Set pivot equal to the right array

Set an element initialized above equal to the lowest point

Create a for loop for the main math of the sorting algorithm:

Setup the compare stats to keep count

Create an if statement so that when the array is less than the pivot it will execute:

Setup the move stats to keep track

Swap the arrays

Create a temp array to avoid changing the array before swapping

Create the main mathematical methods used to swap

Swap the array with the greater element

Store it first in a temporary array to avoid replacing it

Include a math portion for this that swaps it correctly

Returns the variable with a counter

Create a function called q sort:

Initialize the pivot index

Create an if statement so that if the left is less than the right it runs:

Recursive function that inputs the values such as left and right

Create the main quick_sort function:

Create an if statement so that if the elements are below eight:

It'll run shell sort in its place

If not just run q sort which acts as the regular quick sort function

Design Process for shell.c

The shell sort algorithm works by sorting out elements far apart from each other so it moves elements in front until they are all in the right place. Shell sort used gaps to execute its process to produce a sorted array. With the given python pseudocode I was able to figure out a way to translate it into C which I did with the use of two functions and a few if, else, while and for statements.

#include any of the necessary header files

Function for traveling to the next gap:

Initialize any variables used within the function

If statement that iterates if a variable is less than one:

If statement so when the variable is less than or equal to two

This sets a variable equal to one

An else statement otherwise

Setting the variable equal to a given math function

If none of the statements is true then simply set the variable equal to 1 again

Return the variable

Main shell sort function:

Be sure to initialize any variables that will later be used

Create the main shell sort loop of the gaps with a for statement:

Create another for statement for the variable used

Set a variable equal to another

Create a temporary array so the array isn't instantly replaced

Place the compare stats before the while loop for the math

Create a while loop for an element and the array to be compared:

Include the move stat to keep track of the moves

Set the array equal to the given math portion

Set the variable equal to the same as above

Set the array equal to the temp array

Create an if statement to end the for loop if the gap is equal to 1

Break the function

Design Process for sorting.c

My sorting.c file compiled all my previous sorting algorithms into one main file so that they would all be executable in a singular location and accessible through a singular file.