CSE13S Assignment 3 Design

Zion Kang

February 1 2023

1 Shell Sort

Description:
In this sort, we iterate through an array *gaps* to continuously reduce the *gap* between each pair of compared items. For each gap in the gap sequence (array), the function compares the pairs in *arr* that are *gap* indices away from each other. Pairs are swapped if they are in the wrong order.

Pseudocode:
Shell Sort Function:
for each gap in the gap sequence
create a nested for loop that iterates from the gap to the length of the array passed into the function
- set index j as i (iteration)
- create a temp variable holding the value of arr[i]
while value for current index in array is less than the compared value - set current value as compared value
- decrement the index j by the gap (used later to complete swapping of elements)
- set compared value as previously held current value (temp variable from above)

2 Heapsort

Description:
In this sort, we utilize three helper functions to build and fix our heap as well as sort it. The max child function simply returns us the child of a heap with the max value. The fix heap function simply fixes the removal of the largest elements from the heap so as to keep the structure of the heap tree. The build heap function builds us the actual heap tree structure. Finally, the heap sort function itself utilizes these functions to build a heap with the given array then sorts through it by continuously swapping heaps and fixing it (using the fix heap function).

Pseudocode:
max child:
set index for left child
set index for right child
if right child is ¡= last and array value of (right child - 1) ¿ (left child - 1)
return right as max child
otherwise simply return left as max child

fix heap:
set boolean for looping condition (heap to fix)
set mother index as first
set great index as call on max_child (passing in indices for mother and last)

loop while mother is less than or equal to last//2 and found is not true

if mother heap is less than great heap
- swap the two heaps and replace mother with current
- set new great as return of max child between mother and last heap

otherwise (if mother heap is greater than great heap)
- set found as true

build heap:
looping from last heap to first heap, decrementing by 1
call on fix heap function to restructure array into heap (passing in father and last heaps)

heap sort:
set first and last indices (1 and len(array passed in), respectively)
call build heap to create heap (passing in first and last heap indices)

looping through each leaf of heap tree (backwards from last to first, decrementing by 1)
swap current leaf in iteration and first heap
call on fix heap passing in first and current leaf

3 Quicksort

Description: In this sort, we partition the passed in array into two sub-arrays by selecting an element from the array and designating it as a pivot. This pivot serves as a marker for which the elements that are less than the pivot go in the left sub-array and elements greater than or equal to the pivot go in the right sub-array. With these partitioned parts of the array, we recursively call quicksort.

Pseudocode:
partition:
set index for low - high
looping through index j from low to high
if value of current index in array is less than high value
- increment index i
- swap array values for indices i and j
swap array values for i and hi
return i + 1
quicksorter:
if array value of low is less than high
set variable for pivot and call on partition
call quick_sorter on lower partition
call quick_sorter on higher partition
quick sort:
call on quick_sorter with entire array (so as to start the partition)

4 Merge Sort

Description:
In this sort, we implement a similar sorting method to that of Shell Sort. Instead of sorting pairs of elements that are a set gap apart, we k-sort the even and odd subsequences of the array, where k is some power of 2.

Pseudocode:
comparator:
if array value of index x ¿ index y
    swap the two values

batcher sort:
if len of array is 0
- return nothing

create n variable for length of array create t variable for bit length of n create p variable shifting left (t - 1)

loop while p var > 0
- create q variable for shifted left (t - 1)
- create r variable as 0 for now (used later)
- create d variable set as p

loop while d greater than 0 - loop from range 0 to (n - d)
    if (i & p) is equal to r
    - pass in i and (i + d) to comparator function
- set d as (q - p) - set q shifted right 1 - set r as p
set p as shifted right 1

5 Test Harness

Create a test harness for my implemented sorting algorithms. In this test harness, create an array of pseudorandom elements and test each of the sorts.
The test harness must support the following command-line options:
- a: employ all sorting algorithms
- h: enable heap sort
- b: enable batcher sort
- s: enable shell sort
- q: enable quick sort
- r seed: set random seed to *seed* (default seed of 13371453)
- n size: set array size to *size* (default size of 100)
- p elements: print out *elements* number of elements from array (default number of elements to print is 100)
- H: print out program usage

Create separate helper functions that randomize the array (as well as bit mask to fit 30 bits) and print the array in desired format (5 elements a line).

Allocate memory for array and free properly at end so as to prevent memory leak.