Assignment 3 pdf:

Description:

In this assignment, we will be using four sorts Shell sort, Batcher sort, Heap sort, and Quick sort and sort a random array with 15 items. The testing algorithm while having a random array generator function. Then the four algorithms will sort the lists and count the number of swaps and compares.

Files to include:

- batcher.c implements Batcher Sort.
- batcher.h specifies the interface to batcher.c.
- •shell.c implements Shell Sort.
- shell.h specifies the interface to shell.c.
- gaps.h provides a gap sequence to be used by Shell sort.
- heap.c implements Heap Sort.
- heap.h specifies the interface to heap.c.
- quick.c implements recursive Quicksort.
- quick.h specifies the interface to quick.c.
- set.c implements bit-wise Set operations.
- set.h specifies the interface to set.c.
- stats.c implements the statistics module.
- stats.h specifies the interface to the statistics module.
- sorting.c contains main() and may contain any other functions necessary to complete the assignment

Pseudo code:

batcher.c:

- Have a comparator function:
    - Pass two elements into the function
    - If one is larger than the other swap the two elements
- Have a batcher function:
    - Have a variable p that is a value one whose bits are shifted to the left by the length of the array - 1
    - Loop through until p is less than or equal to zero
        - Create a new var d = p
        - Create = 0

        - Create q with that is a value one whose bits are shifted to the left by the length of the array - 1
        - Have a second while loop d > 0
            - Have a for loop that goes through all the elements from d to the end
                - If i and p = r then comparator A[i] and A[i + d]
            - d = q - p
            - Shift q left one
            - Set r = p
        - Shift p left by one

quick.c:

- Have a partition function pass the smallest and largest element, lo and hi respectively
    - Create a var i = lo -1
    - Loop For all the elements between lo and hi
        - Then check if the element is smaller than element bellow hi
            - If true i += 1 and Swap A[i - 1] and A[j - 1]
        - Then swap A[i], A[hi - 1]
    - Return i + 1
- Have a quick sorter function that you pass a list pointer and the variables lo lowest value and hi highest value:
    - if lo < hi:
        - Create p = partition(A, lo, hi)
        - Run quick_sort(A, lo , p - 1)
        - Run quick_sort(A, p + 1 , hi)
- Create a function quick_sort pass the list
    - Run the quick_sorter (A, 1 , len(A))

heap.c:

- Create the function max_child and pass A: list , first : int , last : int
    - Create a var left = 2 * first
    - right = left + 1
    - if right <= last and A[right - 1] > A[left - 1]:
        - return right
    - return left
- Create the function fix_heap and pass A: list , first : int , last : int :
    - found = False
    - mother = first
    - great = max_child (A, mother , last)

        - while mother <= last // 2 and not found :
            - if A[mother - 1] < A[great - 1]:
                - A[mother - 1] , A[great - 1] = A[great - 1] , A[mother - 1]
                - mother = great
                - great = max_child (A, mother , last)
            - Else:
                - found = True
- Create a function build_heap and pass the parameters A: list , first : int , last : int:
    - for father in range (last // 2 , first - 1 , -1) :
        - fix_heap (A, father , last)
- Create a function heap_sort and pass it A: list :
    - first = 1
    - last = len(A)
    - build_heap (A, first , last)
    - for leaf in range (last , first , -1) :
        - A[first - 1] , A[leaf - 1] = A[leaf - 1] , A[first - 1]
        - fix_heap (A, first , leaf - 1)

shell.c:

- Create a function shell_sort and pass it A: list:
    - for gap in gaps :
        - for i in range (gap , len (arr)) :
            - j = i
            - temp = arr[i]
            - while j >= gap and temp < arr[j - gap]:
                - arr[j] = arr[j - gap]
                - j -= gap
            - arr[j] = temp

sorting.c :

This is a getopt file that has case staments that generates a random array and uses case statements to run the appropriate sorting algorithm.