Assignment 3 DESIGN.pdf

Julian Chop

February 4, 2023

1 Brief Description of Assignment

For this assignment, we are tasked with implementing 4 different sorting algorithms that sorts an array in increasing order. We must also implement a test harness for these algorithms and compare them with each other. The algorithms we will be implementing are Shell Sort, Batcher Sort, Heapsort, and Recursive Sort.

2 Shell Sort

- Shell sort is very similar to the insertion sort algorithm. It will sort elements that are far apart, and then sort with a closer gap after every iteration until every element is sorted.

```
def shell_sort(array):
    for gap in gaps: //array of gaps in decending order
        for i in range(gap, length of array):
            j = i:
            temporary = array[i]
            while j >= gap and temporary < array[j - gap]:
                array[j] = array[j - gap]
                j -= gap
                array[j] = temporary
```

3 Heap sort

- a Heap sort is implemented as a specialized binary tree and sorts its elements using 2 routines: building a heap and fixing a heap. First, take the array to sort and build a heap from it. Then, fix the heap by removing the top of the heap and putting it in the end of the array. Then fix the order of the heap, largest elements going to the top.

```
def max_child():
    find biggest element in the heap and return it
```

```
def fix_heap():
    remove top element from the heap
    if heap not in correct order:
        fix heap
def build_heap(array, first, last):
    for father in range(last // 2, first -1, -1):
        fix_heap(array, father, last)
def heap_sort():
    first = 1
    last = len(array)
    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)
```

4 Quick Sort

- Quick sort uses a divide-and-conquer method to sort arrays. It chooses an element and makes it a pivot. Then it splits the array in two, any elements less than the pivot goes to the left sub-array and any elements greater go to the right sub-array. Then it will do this process with the sub-arrays and so on until the whole array is sorted.

```
def partiton(array, lo, hi):
    assign pivot
    for i in array:
        if i < pivot:
            move i left array
def quick_sorter(array, lo, hi):
    if lo < hi:
        p = partition(array, lo,hi)
        quick_sorter(array, lo, hi)
        quick_sorter(array, p + 1, hi)
def quick_sort():
    quick_sorter(array, 1, length(array)
```

5 Batcher's Odd-Even Merge Sort

- Batcher's method is a sorting network. Sorting netwoeks have a fixed number of wires, one for each input. These wires are connected to comparators that compare two wires. If the 2 wires are out of order, they are swapped.

```
def comparator(array, x, y):
    if array[x] > array[y]:
        array[x], array[y] = array[y], array[x]
def batcher_sort (A: list ) :
    if len(A) == 0:
        return

    n = len(A)
    t = n. bit_length ()
    p = 1 << (t - 1)

    while p > 0:
        q = 1 << (t - 1)
        r = 0
        d = p

        while d > 0:
            for i in range (0 , n - d) :
                if (i & p) == r:
                    comparator (A, i, i + d)
            d = q - p
            q > >= 1
            r = p

        p > >= 1
```

6 Sorting.c

- This file will contain main() and the test harness for my algorithms.

```
main():
    default seed = 100
    default size = 100
    default element = 100

    make empty set

    while (getopt()):
        switch opt{
            case a: employs all alg
            case h: enables heap
            case b: enables batch
            case s: enables shell
            case q: enables quick
            case r: set seed
            case n: set lenth of array
            case p: set elements
            case H: set seed
        }
```