Assignment 3 – Sets and Sorting

Mateo Garcia

CSE 13S – Spring 2023

Purpose

The purpose of this program is to allow the user to sort list of random numbers and also have a set of functions to perform set operations. The to-be sorted list will be made up of randomly generated numbers from a seed the user inputs, with a size also inputted from the user. There are 5 different sorts that user can choose: insertion sort, shell sort, heap sort, quick sort, and batcher merge sort. There is also a statistics file that will show the amount of comparisons, and operations, in order to better understand these sorting algorithms. The user can also specify the amount of elements the user wants to print out from the array. As for the sets, the set operations will be used for insertion, removing values, checking for membership, finding union or intersection, or finding the difference or complement. These operations will be used to identify the flags in the command line.

How to Use the Program

By running **./sorting *-[flag1] -[flag2]...*** you can add 10 different options to tell the program which sorting algorithm to use, as well as input the size of your unsorted array, and a seed to fill it with random numbers. There are also flags to print out number of elements of the array, and a help option.

- -a : Employs *all* sorting algorithms implemented
- -i : Enables Insertion Sort
- -s : Enables Shell Sort
- -h : Enables Heap Sort
- -q : Enables Quick Sort
- -b : Enables Batcher Sort
- -r [seed] : Set the random seed to to number specified after -r. The default seed should be 13371453.
- -n [size] : Set the array size to number specified after -n. The default size should be 100.
- -p [elements] : Enable printing of statistics to see computed terms and factors for all tested functions.
- -h : Displays a help message.

Program Design

Data Structures

There are only two big data structures used for this assignment. Sets, and arrays. The arrays will contain the unordered elements to be used for sorting, and will also hold the sorted elements after the algorithms are executed. As for sets, they use 8 bit values that (1 or 0) and denote which operations to execute. Other data structures used are ints for values inside the arrays, and the random function to create the unordered elements in the array.

Algorithms

Each sorting algorithm has it's own algorithm to sort the sorted array.

```
insertion sort algorithm ( arr ):
    n = length(arr)
    for i = 1 to n - 1
       j = i
       while j > 0 and A[j-1] > A[j]
           swap(A[j], A[j-1])
           j = j - 1
       end while
```

```
shell sort algorithm ( arr )
    gap = len(arr)//3
    while gap > 0:
      for start in range(gap):
        insertionSort(arr,start,gap)
      print("After increment of size",gap,
            "the list is", arr)
      gap //= 3
```

```
quick sort algorithm ( alist, first, lastx )
    if first<last:
        splitpoint = partition(alist,first,last)
        quickSort(alist,first,splitpoint-1)
        quickSort(alist,splitpoint+1,last)
}
partition algorithm ( alist,first,last ):
    pivotvalue = alist[first]
    leftmark = first+1
    rightmark = last
    done = False
    while not done:
        while leftmark <= rightmark and alist[ leftmark] <= pivotvalue:
            leftmark = leftmark + 1
        while alist[ rightmark] >= pivotvalue and rightmark >= leftmark:
            rightmark = rightmark -1
        if rightmark < leftmark:
            done = True
        else:
            print(f"swapped to fix pivot {alist[ leftmark]},  {alist[ rightmark]}")
            alist[ leftmark], alist[ rightmark] = alist[ rightmark],alist[ leftmark]
            print(alist)
    print(f"swapping {alist[ first]} and {alist[ rightmark]}")
    alist[ first], alist[ rightmark] = alist[ rightmark], alist[ first]
    print(alist)
    return rightmark
```

```
Heap Sort Algorithm

def max_child(A: list, first: int, last: int):
    left = 2 * first
    right = left + 1
```

```
    if right <= last and A[right - 1] > A[left - 1]:
        return right
    return left

def fix_heap(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

def build_heap(A: list, first: int, last: int):
    for father in range(last // 2, first - 1, -1):
        fix_heap(A, father, last)

def heap_sort(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)
```

```
def comparator(A: list, x: int, y: int):
    if A[x] > A[y]:
        A[x], A[y] = A[y], A[x] # Swap A[x] and A[y]

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
```

Function Descriptions

There are several modules and functions that are part of this program.

The set module will have these functions:

- **Set set_empty(void)** This function is used to return an empty set. In this context, an empty set would be a set in which all bits are equal to 0.
- **Set set_universal(void)** This function is used to return a set in which every possible member is part of the set.
- **Set set_insert(Set s, int x)** This function takes x and inserts into set s.
- **Set set_remove(Set s, int x)** This function removes x from s.
- **bool set_member(Set s, int x)** This function returns a boolean (true or false), whether x is in set s or not.
- **Set set_union(Set s, Set t)** Using the OR operator this function returns all the values (non-duplicate) appear in each sets.
- **Set set_intersect(Set s, Set t)** Using the AND operator this function returns all the values that are shared between the sets.
- **Set set_difference(Set s, Set t)** This function returns returns a set of the elements of set s that are not in set t.
- **Set set_complement(Set s)** This function is used to return the complement of a set.

The sorting algorithms will have their own files and own sorting functions and all their functions can be found in Algorithms.

- **Insertion Sort** will just have the insertion function.
- **Shell Sort** will just have the shell sort function.
- **Quick Sort** will have a recursive function for quick sort, and a partition function. The partition is used to place all the elements less than the pivot in the left part of the array, and all elements greater than the pivot in the right part of the array.
- **Heap Sort** will have a max_child function, fix_heap function, build-heap function, and the heap_sort function itself.
- **Batcher Sort** will have the batcher sort function and a comparator function.

Results

Have not started writing code yet.

Audience: Write this section for the graders. If you completed only part of the assignment, explain that here.

To write this section, use your code according to its intended purpose. Does it successfully achieve everything it should? Is anything lacking? Could anything be improved? Talk about all of that here, and use your code's output to prove it.

Numeric results

You can include screenshots of program output, as I have in Fig. 1.

Figure 1: Screenshot of the program running.

Error Handling

When it comes to handling errors, the Assignment specifications are relatively lax. In this section, describe the errors While the assignment only requires that you be close to epsilon in a certain range,