Assignment 3 DESIGN.pdf

Description of the Program:

This program is a compilation of 4 types of sorting algorithms : Insertion sort, Heap sort, Shell sort, Quick sort. The sorting.c main function sets a seed and a random generator to generate an array of default of 100 elements in sorted order along with stats of each method's amount of time it needed to move and compare.

Files included in directory “asgn3”

1. README.md: A file meant to show how to build and run the program and how the program handles bugs.
2. Makefile : A file that builds the sorting.c program and linked all the sort sources file to main
3. Sorting.c:

   Test harness that produces an array based off command line.
   -a : Employs all sorting algorithms.
   -e : Enables Heap Sort.
   -i : Enables Insertion Sort.
   -s : Enables Shell Sort.
   -q : Enables Quicksort.
   -r seed : Set the random seed to seed. The default seed should be 13371453.
   -n size : Set the array size to size. The default size should be 100.
   -p elements : Print out the number of elements from the array.
   -h : Prints out program usage. See reference program for example of what to print

4. DESIGN.pdf: Describes the entire assignment along with all information regarding the program
5. insert.c implements Insertion Sort.
6. insert.h specifies the interface to insert.c.
7. heap.c implements Heap Sort.
8. heap.h specifies the interface to heap.c.
9. quick.c implements recursive Quicksort.
10. quick.h specifies the interface to quick.c.

11. set.h implements and specifies the interface for the set.
12. stats.c implements the statistics module.
13. stats.h specifies the interface to the statistics module.
14. shell.c implements Shell Sort.
15. shell.h specifies the interface to shell.c.

Source code sorting.c

```
Include all headerfiles of sorts,sets and stats
Char pointer names array as each of the sorts
Initialize set sort()
Enumerate  HEAP, SHELL, INSERTION, QUICK as sorts
main()
    Int p=0 takes care of ensuring that elements isn't ever bigger than size
    Opt = 0
    uint64_t seed = 13371453
    uint32_t size = 100
    uint32_t elements = 100
    uint64_t mask = 0x3FFFFFFF in hex

    while( opt = get opt(arguments)!= -1)
    Switch case using opt
        Case a insert_set all the sorts into the command set
        Case e insert_set HEAP
        Case i insert_set INSERTION
        Case s insert_set SHELL
        Case q insert_set QUICK
        Case n takes argument and turn it into into and assign it to size
        Case p takes argument and turn it into into and assign it to element but limit at 100
        Case r take argument and turn it to seed

    if( (p isn't 1 and size is less than or equal to 100)or (element greater than size)
        Element =size
    Initialize pointer *A as unsigned int with calloc (size,size of unsigned int bit 32)
    Initialize pointer *stats from header file stat with malloc (sizeof (stats))

    for(iterating through Sorts with i)
        If i is in command set
```

```
            Srandom seed
            for (unsigned int x =0 iterate by 1 until x<size)
                A[x] = mask & random()        bit masking random to 30
    bits
        set_sort(i,stats,A,size,elements)

    free(A)
    free(stats)

    Return 0
```

```
Void set_sort(Sorts i, Stats *stats, uint32_t *A, uint32_t size, uint32_t elements)

    if (i == 0)
   heap_sort(stats, A, size)

    f (i == 1)
   shell_sort(stats, A, size)

     if (i == 2)
   insertion_sort(stats, A, size)

    if (i == 3)
   quick_sort(stats, A, size)
Print out elements moves and compares of stats

reset(stats)

for(iterating through elements with x) print out columns of 5 from the array A[x]
```

Source code heap.c:

```
Include insert.h to link to main
Include math ,studio stdbool,stdlib

Int max_child(Stats *stats, uint32_t *A, int first, int last)

    int left = 2 * first; int right = left + 1;
    if (right <= last and cmp(stats, A[right - 1], A[left - 1]) == 1) {
        return right;
    }
    return left;

void fix_heap(Stats *stats, uint32_t *A, int first, int last) {
    bool found = false;
    int mother = first;
    int great = max_child(stats, A, mother, last)
    While mother <= integer deviation last not found
        if A[mother - 1]<[great - 1] {
            Swap A[mother - 1] A[great - 1]
            mother = great;
            great = max_child(stats, A, mother, last)
        else
            found = true
void build_heap(Stats *stats, uint32_t *A, int first, int last)
    for (father = integer division last/2 iterate by -1 until father > first - 1)
        fix_heap(stats, A, father, last)

void heap_sort(Stats *stats, uint32_t *A, uint32_t n)
    int first= 1 int last = n
    build_heap(stats, A, first, last);
    for (int leaf = last iterate by -1 until leaf > first)
        Swap A[first - 1], A[leaf - 1])
        fix_heap(stats, A, first, leaf - 1)
```

Source code insert.c

Include insert.h stdio and stdlib

```
Void insertion_sort(Stats *stats, uint32_t *A, uint32_t n)
            For unsigned int i iterate by 1 to n
            unsigned int j equal 1
            unsigned int temp = A[i] while stats moves
            while(j>0 and A[j-1] < temp)stat cmp = 1
                        A[j] equals A[j -1]
                        j -= 1
```

```
      A[j] = temp (stats move)
```

Source code shell.c

```
Gaps function takes unsigned int n
      Sets up a static iter_val = 0
      If iter_val <= 0
            Iter_val = log(3+2*n)/log(3)
      Else
            Iter_val -=1
      return((3^iterval)-1 )/2
shell_sort(takes stats A and n)
      iteration = log(3 + 2 * n) / log(3))
  for iterate again through interaction variable
    gap = gaps(n)

    for ( j = gap; j < n; j += 1)
      h = j
      temp = move(stats, A[j]);
      while ((h >= gap) and cmp(temp, A[h - gap]) = -1
        A[h] = move(stats, A[h - gap])
        h -= gap
      A[h] = move(stats, temp)
```

Source code quick.h:

```
int partition(Stats *stats, uint32_t *A, int lo, int hi) {
 This function interates through the right and left of the center value and swaps them
accorindly
void quick_sorter(Stats *stats, uint32_t *A, int lo, int hi)
  if (lo < hi)
    int p = partition(stats, A, lo, hi)
    quick_sorter(stats, A, lo, p - 1)
    quick_sorter(stats, A, p + 1, hi)


void quick_sort(Stats *stats, uint32_t *A, uint32_t n) {
  quick_sorter(stats, A, 1, n);
```

Notes on the Pseudocode:

The majority of the bases where based of Christian’s section and the provided pseudocode in the assignment doc

The test harness main function was mostly the base cases where based off Eugene’s section

The stats class is constantly being tracked through the sort functions and and each move,cmp,swap will consistently add towards the total for the