Bubble.c

```
Loop from 0 to n_elements -1; variable is i

        Swapped = false;

        Loop from n_elements -1 down to i; decrement counter

                //If current value is smaller than value to the left, swap them!
                If arr[j] < arr[j-1]
                        Swap the above values
                        Swapped = true;

        If swapped = false
                Break as no changes occurred so array must be sorted
```

Shell.c

```
Int next_gap(int n)
        Return 5*n/11
        If n<=2 then return 1
        If n=1, return 0

Void shell(Stats, arr, n_elements)

        Loop while current gap > 0, nextgap is found through the function above

                Loop from gap to n_elements

                        Int j = i
                //perform swap if elements are out of order
                While( j >= step and arr[i] < arr[j-step]
                        Arr[j] = arr[j-step]
                        j-= step
                        Arr[j] = temp
```

quick.c

```
Void quicksort(stats, arr, n_elements)

If n_elements < 8
        //Just shellsort small array as quite efficient at small values
        shellsort(arr)



//Choose pivot that will hopefully split the data a bit evenly
Choose pivot, arr[0] + arr[n_elements/2] + [n_elements-1] / 3

Create 3 arrays, Lower Mid and Upper

Put all values that are lower/higher or equal the pivot in their respective arrays

quicksort(lower)
quicksort(upper)

Rebuild arr with Lower + Mid + Upper
```

Heap.c

```
//General Heap upkeep functions
//Relations between a parent node and its children based on their indexes
L_child
        Return 2n+1
Child
        Return 2n+2
Parent
        Return (n-1)/2
```

```
Up_heap
        //If parent is bigger than current value, swap since we want a minheap
        While (n>0) a[n] < a[parent]
                Swap a[n] and a[parent]
                N = parent;


Down_heap
        While l_child(n) < heap_size

                //Checking that this element is the last in the heap, if it is then it wont have
a sibling
                //Make sure we are not out of bounds
                If r_child == heap_size
                        smaller = l_child

                Else
                        smaller = l_child or r_child, whichever is smaller


                If a[n] < a[bigger]
                        Swap a[n] and a[smaller]
                n=smaller




Build_heap(arr)

        Create new arr called heap
        Loop through all of arr; counter is i
                Heap[i] = a[n]
                Upheap


Heap_sort

Loop through all elements in heap
        //Taking the element from the top of the heap and then moving it to the last index
```

//This index is then ignored for the rest of the loop effectively removing it from the heap

Sorted[n] = heap[0]

Heap[0] = heap[n_elements-n-1]

Down_heap

Sorting.c

PrintArrayI() function

Loop through elements and print given array in specific syntax

Int main(int argc, char **argv)

Set default values, seed = 13371453, size = 100, elements = 100

Create set keys for each value

- Shell = bit 1
- Bubble = bit 2
- Quick = bit 3
- Heap = bit 4
- Usage = bit 5

Loop through all inputs

Switch

Case ‘a’

Enable all sorting methods by setting bits 1,2,3,4 to 1

Case ‘s’

Enable shell by setting bit 1 to 1

Case ‘b’
    Enable bubble by setting bit 2 to 1
Case ‘q
    Enable quick by setting bit 3 to 1
Case ‘h’
    Enable heap by setting bit 4 to 1

Case ‘r’
    Set seed to input
Case ‘n’
    Set size to input
Case ‘p’
    Set printed elements to input
Case ‘H’
    Enable usage by setting bit 5 to 1
Default:
    Print usage and return 1;

Check for invalud inputs

If bit 5 is 1
    Print usage statement

Set seed to (seed)
Create array of inputted length using values from mtrand_64

If bit 2 is 1
    Do bubble sort

If bit 4 is 1
    Do heap sort
If bit 3 is 1
    Do quick sort
If bit 1 is 1
    Do shell sort