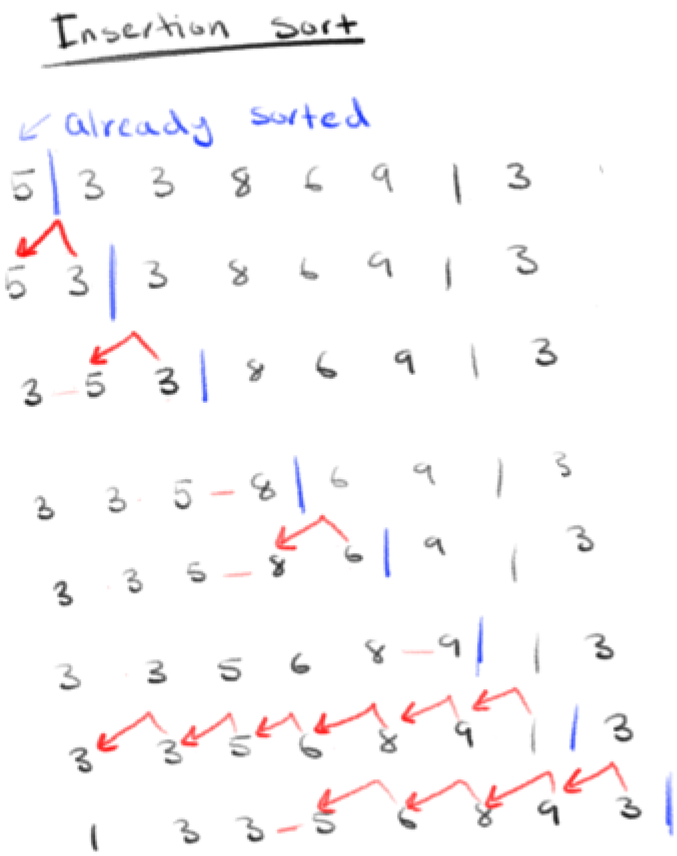
Description of Program:
Do the following sorts, Insertion, Heap, Quick, and Batcher.

Structure:
“.h” files *insertion, batcher, heap, quick* create the functions that will be defined in the “.c” files.
Stats.{h, c} create and define the functions, *cmp, move, swap,* and *reset*. And will keep track of the moves and comparisons used for later outputs.

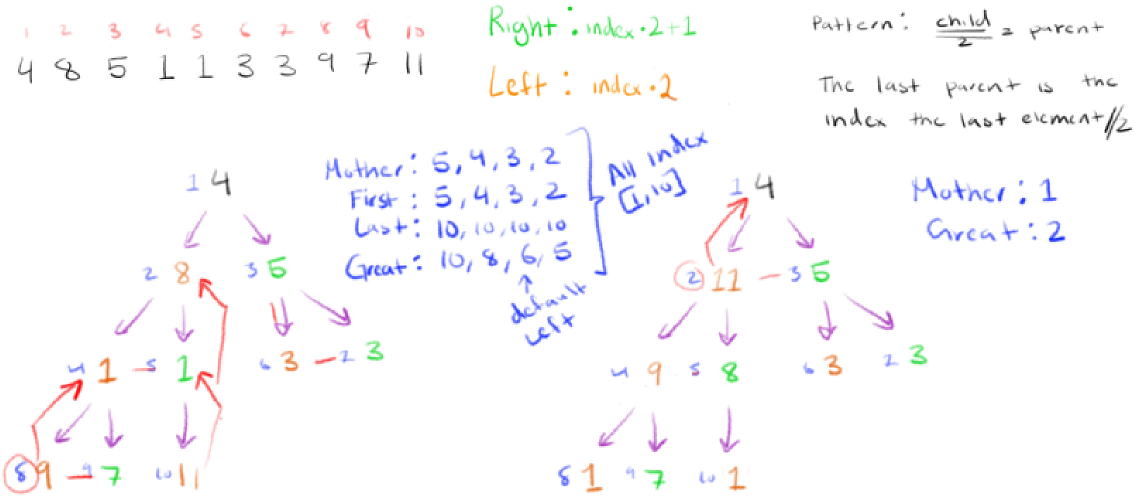
Insertion Sort Function:



A list with 1 element is already sorted so it is the default.
Move the | right with now 2 elements.
If e[1] > e[2], swap
Move | right 3 elements.
If e[2] > e[3] swap.
  If e[1] > e[2] swap.
else nothing
…
continue…

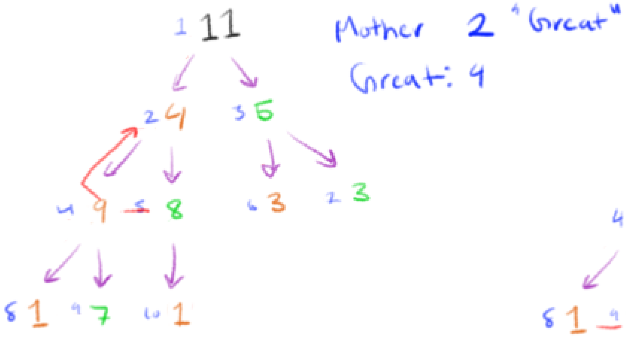
Heap Sort Function:

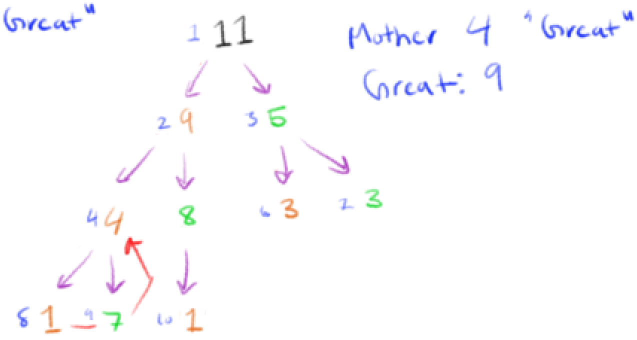
Built Heap:

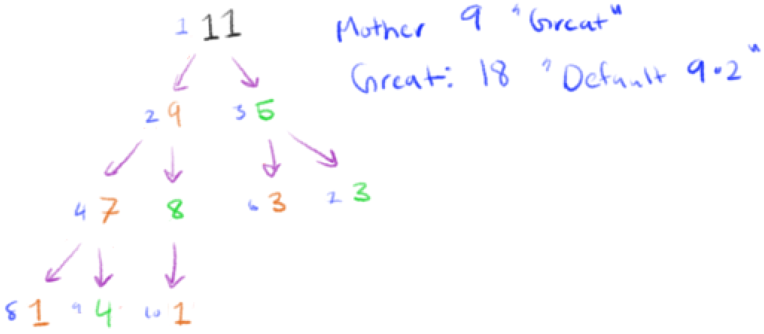


Label the indexes [1,10] because 0*2 is 0.
The pattern noticed is that the parent of any number is the indexof(itself) // 2.
The last index is 10. 10//2 == 5.
A properly built Max Heap means that the parent must always have a greater value than the child.

L[5] == 1 has child L[10] == 11 No sibling L[11].
Great = 10. #No sibblings
Compare (Parent, Child), L[5] < L[10] returns -1
Move down the mother. 5 - 1 == 4.
L[4] == 1 has 2 children L[8] == 9 L[9] == 7. Compare(L[8], L[9]). L[8] > L[9] returns 1.
Great = 8. #Highest value sibbling
Compare(L[4], L[8]), returns -1. Swap.
L[3] == 5 L[6] == 3 L[7] == 3. Compare(L[6], L[7]) Equal, returns 0.
Great = 6. #Default
Compare(L[3], L[6]), returns 1. Nothing.

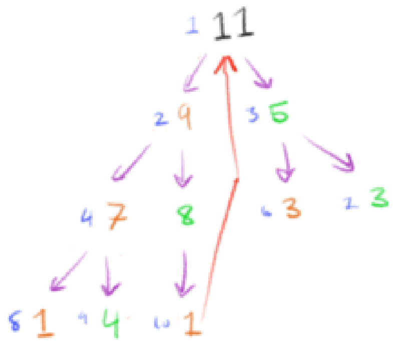






Mother > $\frac{Last}{2}$ Therefore mother is sad and has no children

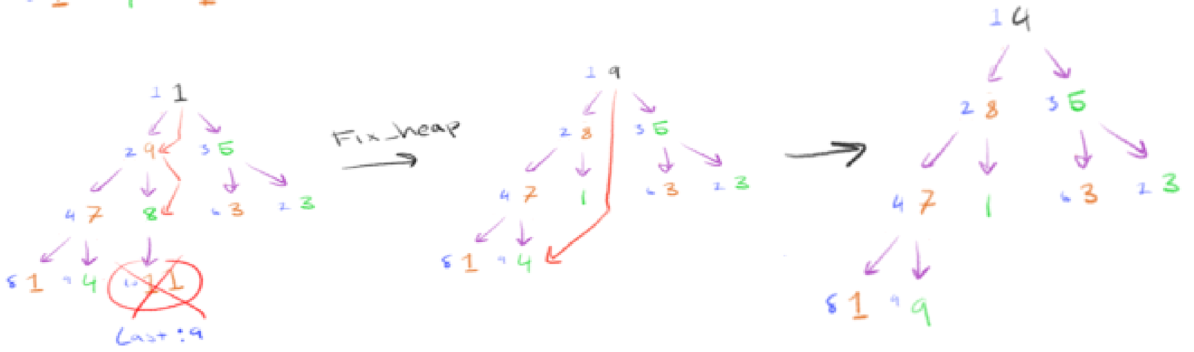
Heap Sort Function
Swap A[1] and A[Last]
A[Last] is now greatest because A[1] if a Max Heap was greatest.
The greatest number is A[Last] now move down 1.
Last = Last - 1
Fix heap. Make it back into a Max Heap so A[1] is the greatest of A[1 to Last]
Swap A[1] and A[Last]



Parent is always greater than the child

just like in real life

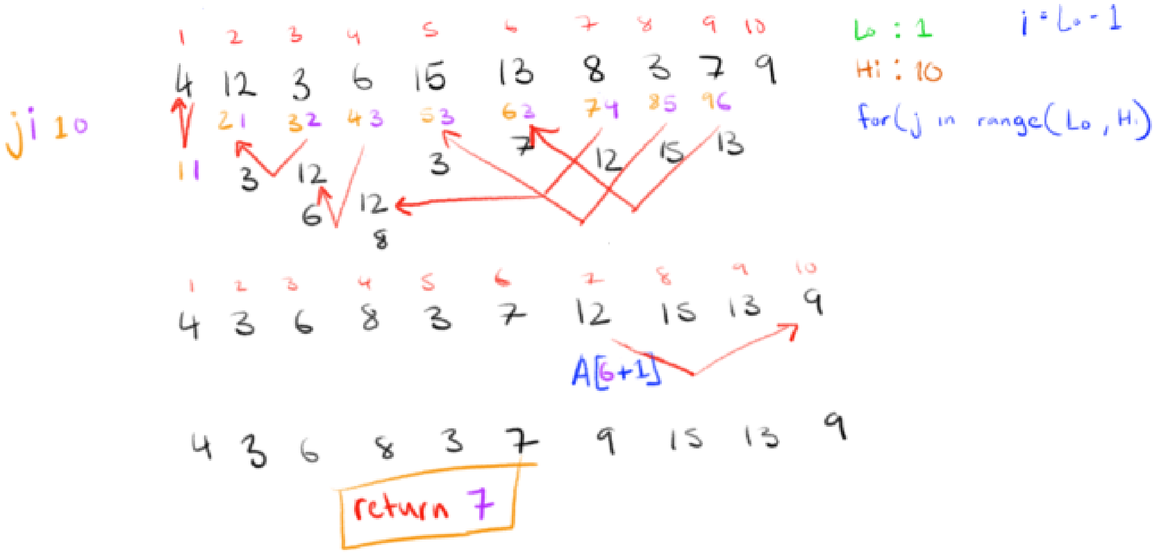
[1 9 5 7 8 3 3 1 4|11] Last: 10 → 9



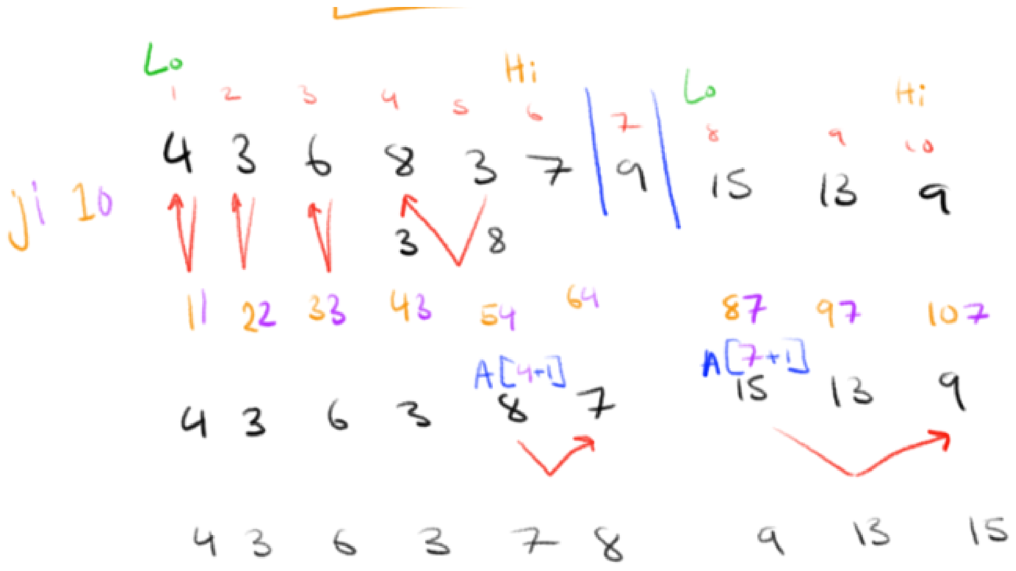
keep repeating till it is ordered

Quick Sort Function:

Partition Function:



Quick Sorter:



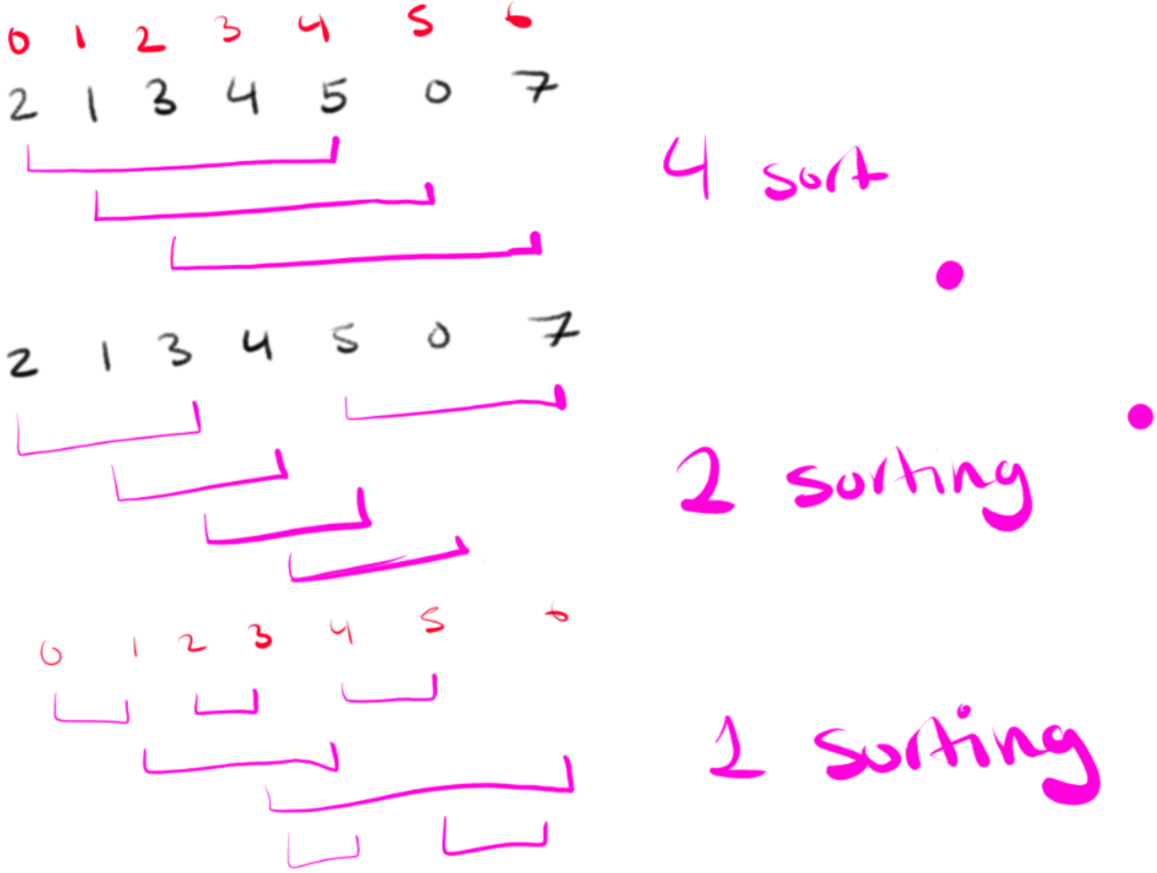
Partition function - Finds the location where to partition

- It determines the location by using a variable j to increment through array and variable i to increment under the condition that the value A[j] is less than the maximum value, then it swaps with the A[i].
- Variable i will eventually lag behind j as the array encounters values higher than A[last].
- At the end swap the A[last] with A[i]. Logically if A[last] is greater than i elements in the list then it should be at position i in a sorted array

Quick Sorter - calls the Partition over and over from Partitions

- (Lo, Partition - 1)
- (Partition + 1, Hi)
- Exclude Partition because it is already in its correct index. By calling it over and over until Lo == Hi or the length of the section is 1, everything will become a partition and order itself.

Batcher’s Odd-Even Merge Sort Function:



0	1	2	3	4	5	6
2	1	3	4	5	0	7

——4 Sorting——

Compare(L[0], L[4]), return -1. Nothing.

Compare(L[1], L[5]), return 1. Swap.

0	1	2	3	4	5	6
2	0	3	4	5	1	7

Compare(L[2], L[6]), return -1. Nothing.

——2 Sorting——

Compare(L[0], L[2]), return -1. Nothing.

Compare(L[4], L[6]), return -1. Nothing.

Compare(L[1], L[3]), return -1. Nothing.

Compare(L[2], L[4]), return -1. Nothing.

Compare(L[3], L[5]), return 1. Swap.

0	1	2	3	4	5	6

```
2       0       3       1       5       4       7

——1 Sorting——
Compare(L[0], L[1]), return 1. Swap.
0       1       2       3       4       5       6
0       2       3       1       5       4       7
Compare(L[2], L[3]), return 1. Swap.
0       1       2       3       4       5       6
0       2       1       3       5       4       7
Compare(L[4], L[5]), return 1. Swap.
0       1       2       3       4       5       6
0       2       1       3       4       5       7
Compare(L[1], L[4]), return -1. Nothing.
Compare(L[3], L[6]), return -1. Nothing.
Compare(L[3], L[4]), return -1. Nothing.
Compare(L[5], L[6]), return -1. Nothing.

0       1       2       3       4       5       6
0       2       1       3       4       5       7
```