RAVELO, JOSHUA S.

OTHER SORTING ALGORITHMS

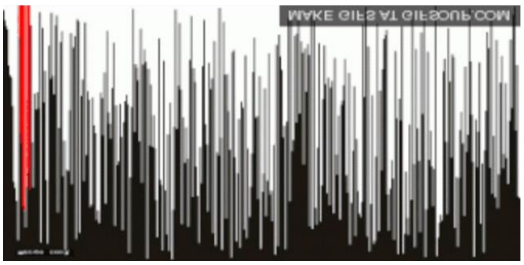
Merge Sort

-O(n log(n))

-Divide and Conquer Algorithm

-Stable

-Continuously divides an array into two halves, recurses on both the left subarray and right subarray then merges the two sorted halves.



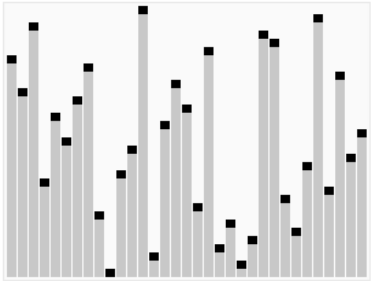
Heap Sort

-O(n logn)

-Build a heap first (parent node greater than child/children)

-Once built, removes the root node from the heap and places it at the end of the sorted array (while doing so, it repairs the heap)

-Efficient and Stable



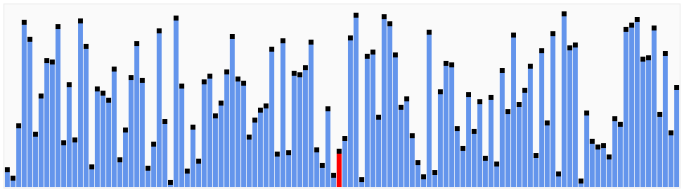
Quick Sort

-O(n log(n))

-Also, a Divide and Conquer Algorithm

-Partitions the input array into 2 subarrays; all left subarray are less than or equal to all elements in the right subarray (recursively sorts two subarrays)

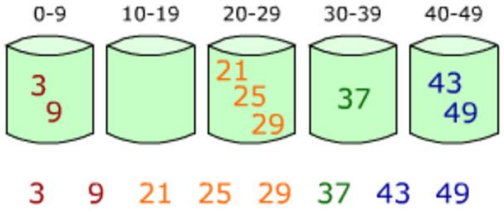
-Efficient and Stable



Bucket Sort

-Ω(n + k)

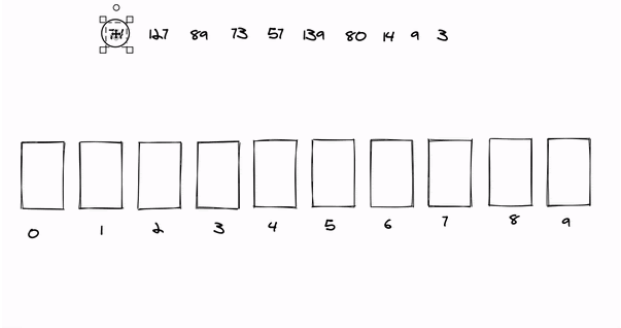
-A sorting method that divides an array's elements into a number of buckets. The buckets are then individually sorted, either using a different sorting algorithm or by recursively applying the bucket sorting process.



Radix Sort

-Ω(nk)

-A sorting algorithm that, like bucket sort, divides an array's items into a number of buckets. Radix sort, on the other hand, varies from bucket sort by ‘re-bucketing' the array after the initial run, as opposed to sorting each bucket and combining the results.



Merge Sort:

```c
void merge(int arr[], int left, int mid, int right)
```

This function is a part of the Merge Sort algorithm. It takes an array `arr` and three indices (`left`, `mid`, `right`) to merge two subarrays. The `merge_sort` function recursively divides the array into two halves until each subarray has only one element. Then, the `merge` function is called to merge these sorted subarrays.

Heap Sort:

```c
void heapify(int arr[], int n, int i)
```

This function is a part of the Heap Sort algorithm. It converts the array into a max heap, where the value of each node is greater than or equal to the values of its children. The `heap_sort` function first builds a max heap, then repeatedly extracts the maximum element from the heap and places it at the end, effectively sorting the array.

Quick Sort:

```c
int partition(int arr[], int low, int high)
```

This function is a part of the Quick Sort algorithm. It chooses a pivot element and partitions the array around the pivot, such that elements smaller than the pivot are on the left, and elements greater than the pivot are on the right. The `quick_sort` function recursively applies this partitioning process to sort the array.

Main Function:

```c
int main()
```

In the `main` function, the user is prompted to choose one of the sorting algorithms (Merge Sort, Heap Sort, or Quick Sort) or exit the program. After choosing an algorithm, the user inputs the number of elements and the actual elements. The selected sorting algorithm is then applied to the input array, and the sorted array is displayed. This process repeats until the user chooses to exit.

Each sorting algorithm also prints the intermediate state of the array after each pass for better understanding of the sorting process.

Overall Flow:

1. The user is prompted to choose a sorting algorithm or exit.
2. The user inputs the number of elements and the actual elements.
3. The selected sorting algorithm is applied to the input array.
4. The intermediate states of the array during the sorting process are displayed.
5. The sorted array is displayed at the end.
6. Steps 1-5 are repeated until the user chooses to exit.

The use of dynamic memory allocation (`malloc` and `free`) is also notable to handle arrays of variable size entered by the user.