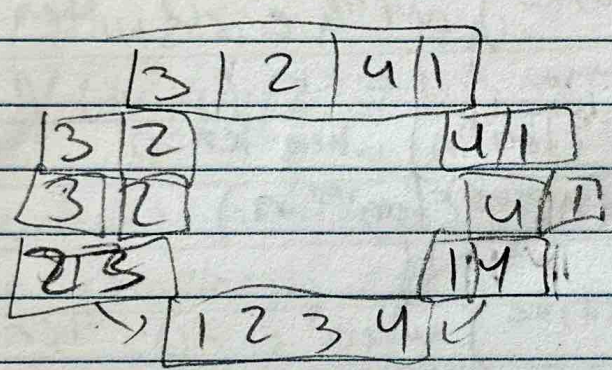
MergeSort

```
mergesort (arr[], l, r)
  if l<r
     mid = (l+r)/2          Break array in half
     merge sort(arr, l, mid)
     mergeSort(arr, mid+1, r)
     merge(arr, l, mid, r)
```



If there are odd number of elements, Left half gets extra element

The merge operation

Scan through both arrays at same time
Compare first elements
Insert smaller element in return array
If one of the arrays runs out, add remaining elements from other array

- Both arrays for the merge operation are already sorted.

- Arrays of size 1 are already sorted

```
Merge(A, B, m, n){
    i, j, k = 0
    while (i <= m && j <= n) {
        if (A[i] <= B[j]) {
            C[k++] = A[i++]
        }
        else{
            C[k++] = B[j++]
        }
    }
    while (i <= m) { C[k++] = A[i++] }
    while (j <= n) { C[k++] = B[j++] }
    return C
```

MergeSort Time Complexity

merge function has 3 while loops but none are nested.

This means time complexity of merge function is $\Theta(n)$

$T(N) = 2T(N/2) + \Theta(n)$

Time complexity: $\Theta(n \log n)$

MergeSort is Stable
- equivalent elements retain their relative positions after sorting

Space Complexity $\Theta(n)$

Quicksort

```
quicksort(arr[], l, r)
if l < r;
  pivot = partition(A, l, r)        Break array in half
  quickSort(arr l, pivot_index - 1)  quicksort left half
  quicksort (arr, pivot_index + 1, r) quicksort right half
```

Partition operation

- Assign a Pivot
- Compare elements with pivot, from left to right
- Elements less than/equal to pivot should be swapped to the left.
- Swap pivot to proper position

- Quicksort is unstable.
- Worst case, Quicksort is $\Theta(n^2)$ time
- Worst case, Quicksort is $\Theta(n)$ space.
- Best case, Quicksort is $\Theta(n \log n)$ time

Heaps

Heaps are complete binary trees which have some particular properties

Complet Binary tree thumb rule:
The leaf nodes should be present from left to right (no blanks in the middle)

Heaps have the following property:

1. The root is greater than (or equal to) its children (max heap)

2. The root is lesser than (or equal to) its children (min heap)

As long as both children are less than/equal to the parent, it doesn't matter whether they are right/let child. (max heap)

We can have larger element further down in the heap as long as they are not descendants of an element that is smaller.

We can represent heaps in array form storing elements from top to bottom, left to right.

i's left child = 2i+1
i's right child = 2i+2
i's parent = (i-1)/2

To preserve the complete binary tree condition, we insert new nodes from left to right

When incoming value is greater than the parent we can swap it with parent until parent is greaterthan or equal to incoming value.

Time complexity for inserting into a heap

Best case: $\Theta(1)$
Average case: $\Theta(1)$
Worst case: $\Theta(\log n)$

Build Heap Time complexity
worst case: $\Theta(n)$
- $n/2$ nodes at height 0
- $n/4$ nodes at height 1
- $n/8$ nodes at height 2
- $n/2^h$ nodes at height h

Heapsort.
1. Build a heap from your elements
2. Delete all elements from top

Buildheap: $O(n)$
Delete: $O(\log n)$, for each of the n elements
Heap sort: $O(n \log n)$

HeapSort is:
- in place: does not need extra space.
- unstable: equivalent elements may be swapped.
- Slower than quicksort in practice

Counting Sort

1. Count the number of unique elements with an occurence counter for each

2. Make a new array by placing elements however many times it occurs in the input array

This is unstable
For stable:

1. Count the elements (same as first)

2. Update counts to be cummulative

3. Traverse array in reverse order, putting references in the right place by referencing our count table.

Count table acts as index pointer (count-1)

Time complexity
Array traversal is $\Theta(n)$
Assigning k counters for k values is $\Theta(k)$ time
$\rightarrow \Theta(n+k)$
Space complexity $\rightarrow \Theta(n+k)$

For $k = n$, time / space complexity is $\Theta(n)$

Counting sort is very space expensive

Counting sort is only useful for arrays with lots of repeating elements

Radix sort used to sort punch cards

Bucket sort is modified Counting sort

They have same issues as counting sort.

Stalin Sort

- Traverse the array