MERGE SORT

TYPE OF DIVIDE AND CONQUER ALGORITHM

1) Set the problem up
2) Break the problem into smaller subproblems.
3) Solve the subproblem by continuing to further break the subproblem into smaller problems recursively until the subproblems reach a state where they cannot be broken down any further, aka the base case, and solve the smallest subproblems
4) Begin recombining the solutions to the subproblems until one reaches back up to the original problem and returns the completed solution.

THE SORTING PROBLEM

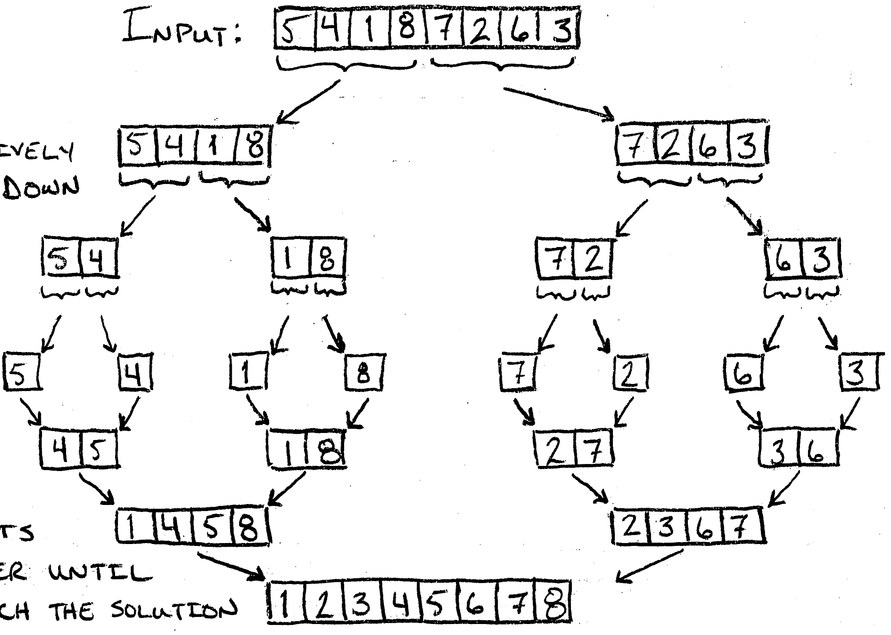
A class of problems in which you receive an input of unsorted elements (numbers, objects, etc) and you want the output to be arranged in a specific order.

INPUT: [5, 4, 1, 8, 7, 2, 6, 3]

OUTPUT: [1, 2, 3, 4, 5, 6, 7, 8]

(con't on next sheet)

Merge Sort Example



Recursively break down to base case

Base case

Merge sorted elements together until you reach the solution

Base case: A single number is in sorted order

Pseudocode for Merge Sort

```
1. Receive the input array, P, of n elements
2. Find the integer halfway point given the length of the array, P/2
3. Create new subarrays from P[Ø] up to, but not including
   P[P/2] and P[P/2] to P[n-1] (zero-based index)
4. Repeat steps 2 and 3 until you reach the base case.
   Base case: a single number or no number is sorted.
5. Merge sorted arrays by comparing items in each sorted array and
   placing the smaller of the two items in a combined array to sort
   the larger array.

   Let C = output array of length m
   Let A = 1st sorted array (m/2)
   Let B = 2nd sorted array (m/2)
   Let i = index position in array A.
   Let j = index position in array B.
   Let k = index position in array C.
```

(con't on next sheet)

PSEUDOCODE FOR MERGE SORT (con't)

5. MERGE STEP (con't)

```
FOR k=Ø TO k=m-1
    IF A[i] < B[j]
        C[k] = A[i]
        INCREMENT i
    ELSE
        C[k] = B[j]
        INCREMENT j
```

While going through this loop we could exhaust one sorted array before the other. A case to check to see if we have reached the end of an array can be added and in which case we would just copy the elements remaining into the output array since the elements are already sorted.

6. CONTINUE DOING STEP 5 UNTIL ALL ELEMENTS HAVE BEEN MERGED INTO THE SOLUTION.

ANALYSIS OF MERGE SORT

CLAIM: For every input of n elements, Merge Sort produces a sorted output array and uses at most $(5n)\lg(n) + 5n$ operations.

Use recursion tree analysis, that is, find how many levels of recursion are needed to reach the base case. Using the merge sort example on page 2 for an 8 element array 4 recursive calls were needed to reach the final base cases where no more recursion is necessary. For a binary tree in a more general case, there will need to be $\lg(n) + 1$ levels of recursion.

LET j = HOW FAR DOWN (WHAT LEVEL) WE ARE ON THE RECURSION TREE.

At each level since the original array size is halved and the number of arrays doubles:

NUMBER OF SUBPROBS AT LEVEL $j = 2^j$

NUMBER OF ELEMENTS IN EACH SUBPROB AT LEVEL $j = \frac{n}{2^j}$

TOTAL NUMBER OF OPERATIONS FOR MERGE SORT

From the pseudocode merge step (step 5) lets say there are 5 operations (5 lines of code executed) per loop. Therefore, at level j

OPERATIONS @ LEVEL j = (# OF SUBPROBLEMS)[(OPS/SUBPROBLEM)(SUBPROBLEM SIZE)] →

OPERATION @ LEVEL $j = 2^j\left[(5)\left(\frac{n}{2^j}\right)\right] = 5n$, # OF OPERATIONS INDEPENDENT OF j

TOTAL # OF OPERATIONS = # OF LEVELS (# OPERATIONS/LEVEL) →
$= (\lg n + 1)(5n) \rightarrow \underline{\underline{5n\lg n + 5n}}$

PYTHON MERGE SORT

```
# mergeSort.py
# An example program of the merge sort algorithm.

def main():
  A = [5, 4, 1, 8, 7, 2, 6, 3]
  A = divide_and_conquer(A)
  print("The sorted array is", A)

def divide_and_conquer(input_list):
  # Base case: A single element list or an empty list is sorted.
  if len(input_list) < 2:
    return input_list

  a = divide_and-conquer(input_list[0:len(input_list)//2])
  b = divide_and-conquer(input_list[len(input_list)//2:])

  sorted_list = merge-sort(a, b)

  return sorted_list

def merge_sort(a, b):
  sorted_list = []
  sorted_list_length = len(a) + len(b)
  # Initialize indexes for sublists a and b to use in assignment to
  # sorted-list
  i, j = 0, 0

  for _ in range(sorted_list_length):
    # Ensure we do not run off the end of the list.
    if i >= len(a):
      while j < len(b):
        sorted_list.append(b[j])
        j += 1
      return sorted_list
    elif j >= len(b):
      while i < len(a):
        sorted-list.append(a[i])
        i += 1
      return sorted_list

```

(con't on next sheet)

Python Merge Sort (con't)

```
        if a[i]<=b[j]:
            sorted_List.append(a[i])
            i += 1
        else:
            sorted_list.append(b[j])
            j += 1

    return sorted_List

if __name__ = "__main__":
    main()
```

Go Merge Sort

```
package main
// An example of the merge sort algorithm

import "fmt"

func main() {
  A := []int {5, 4, 1, 8, 7, 9, 2, 6, 3}
  A = divideAndConquer(A)
  fmt.Printf("The sorted slice is %v.", A)
}

func divideAndConquer(xi []int) []int {
  // Base case: A single element or empty slice is sorted.
  if len(xi) < 2 {
    return xi
  }

  a := divideAndConquer(xi[0 : len(xi)/2])
  b := divideAndConquer(xi[len(xi)/2:])

  sortedSlice := mergeSort(a, b)

  return sortedSlice
}

func mergeSort(a, b []int) []int {
  mergedSlice = make([]int, len(a) + len(b))

  // Initialize indices for arrays a and b.
  var i, j int

  for k := 0; k < len(mergedSlice); k++ {
    // Check if we've reached the end of a slice and copy
    // remaining elements from the other slice.
    if i >= len(a) {
      for j < len(b) {
        mergedSlice[k] = b[j]
        j++
        // Increment k as we won't break out of the inner loop.
        k++
      }
      break
```

(con't on next sheet)

Go Merge Sort (con't)

```
    } else if j >= len(b) {
      for i < len(a) {
        mergedSlice[k] = a[i]
        i++
        // Increment k as we won't break out of the inner loop
        k++
      } break
    }

    // Merging the slices
    if a[i] <= b[j] {
      mergedSlice[k] = a[i]
      i ++
    } else {
      mergedSlice[k] = b[j]
      j ++
    }
  }

  return mergedSlice
```