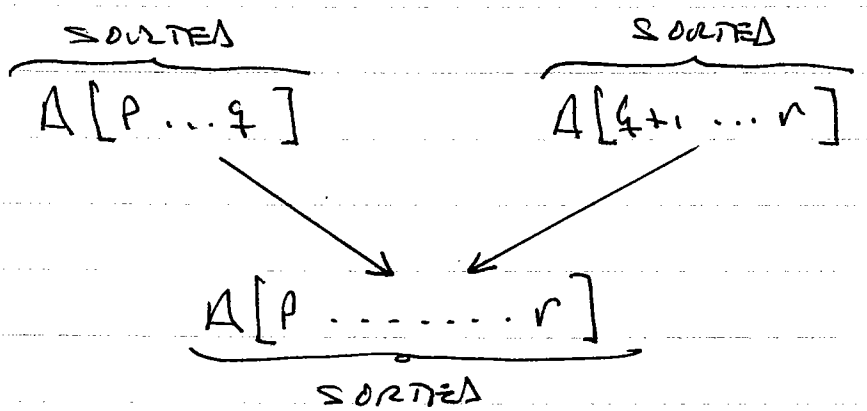
(2.3) Divide & Conquer Algorithms

We often consider Recursive Algorithms, (Also Divide & Conquer). These are Algorithms which call themselves in the course of execution.

MergeSort is a Recursive Sorting Algorithm. It uses a sub-algorithm called Merge. The call Merge(A, p, q, r), where $p \le q < r$, combines sorted sub arrays $A[p \ldots q]$ and $A[q+1 \ldots r]$ into a single sorted sub-array $A[p \ldots r]$



Exercise

Write Pseudo-code for Merge(A, p, q, r), (or Read it on p. 29). Show that Merge Does $r - p$ comparisons to sort $A[p \ldots r]$.

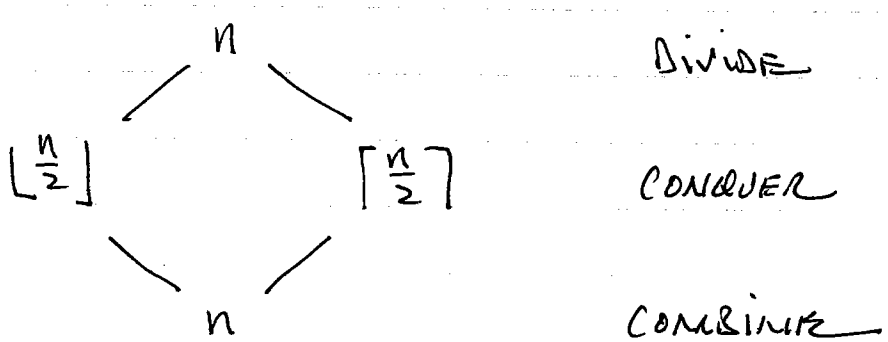
MergeSort (A, p, r)

1.) if $p < r$
2.) $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3.) MergeSort (A, p, q)
4.) MergeSort (A, q+1, r)
5.) Merge (A, p, q, r)

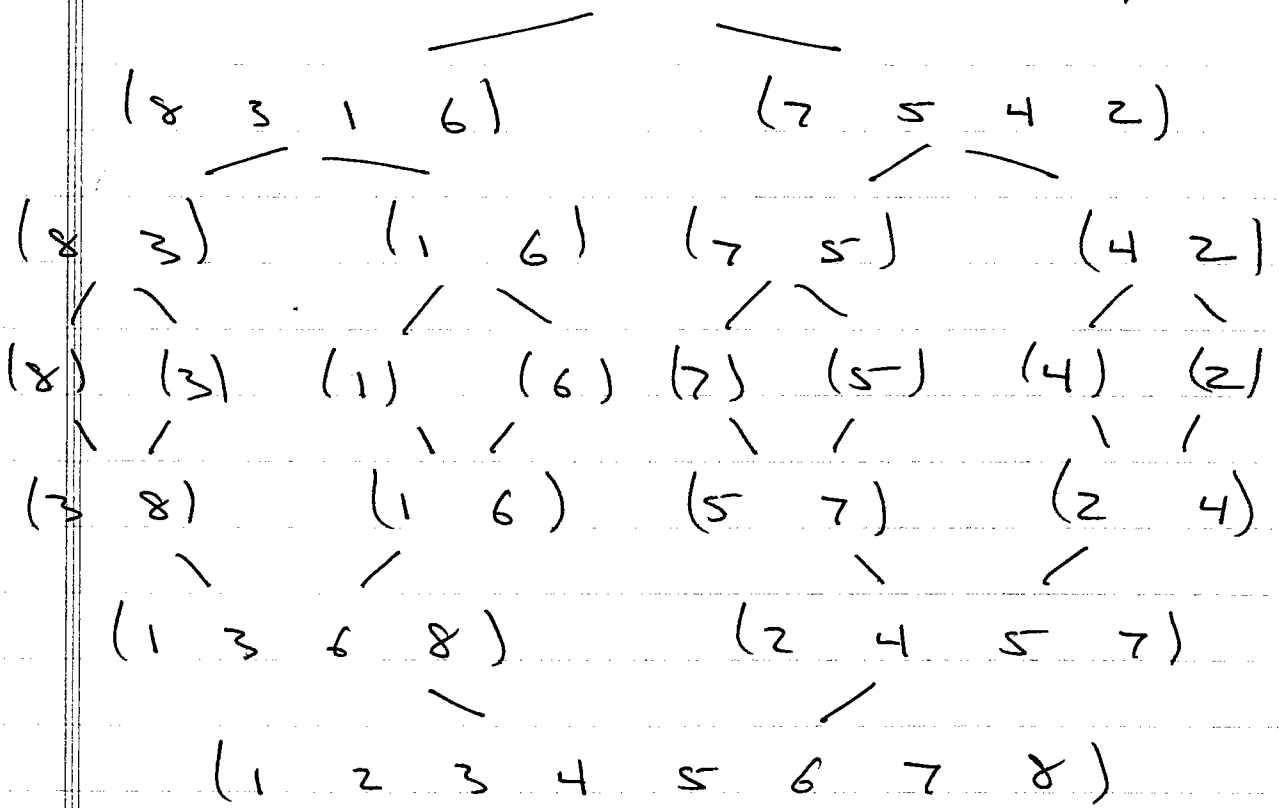
IF $p \geq r$ THEN $A[p..r]$ CONTAINS AT MOST ONE ELEMENT, AND IS THEREFORE ALREADY SORTED. MergeSort DOES NOTHING IN THIS CASE. IF $p < r$ MergeSort IS CALLED RECURSIVELY ON THE SUBARRAYS $A[p...q]$ AND $A[q+1...r]$, WHERE $q = \lfloor \frac{p+r}{2} \rfloor$ IS THE 'MIDDLE' INDEX. THE TWO SORTED SUB-ARRAYS ARE THEN COMBINED USING MERGE.

THE TOP LEVEL CALL MergeSort (A, 1, n) SORTS THE FULL ARRAY $A[1...n]$.

MergeSort DIVIDES THE ARRAY OF SIZE n INTO TWO SUBARRAYS OF SIZE $\lfloor \frac{n}{2} \rfloor$ AND $\lceil \frac{n}{2} \rceil$ RESPECTIVELY.



Ex (8 3 1 6 7 5 4 2)



We wish to determine the runtime of MergeSort. As before we take the comparison operation as barometer.

Let $T(n)$ denote the number of array comparisons performed by MergeSort on arrays of length n. Then

$$T(n) = \begin{cases} 0 & n = 1 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + (n-1) & n \geq 2 \end{cases}$$

Note: Best, Worst, Average are same.

IF n HAPPENS TO BE A POWER OF 2, THIS REDUCES TO

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T\left(\frac{n}{2}\right) + (n-1) & n \geq 2 \end{cases}$$

THE SOLUTION TO THIS RECURRENCE IS

$$T(n) = n \lg n - n + 1 \qquad (n \text{ A POWER OF } 2).$$

HERE $\lg n$ DENOTES $\log_2 n$.

CHECK:

$$\begin{aligned} RHS &= 2T\left(\frac{n}{2}\right) + (n-1) \\ &= 2\left(\frac{n}{2}\lg\left(\frac{n}{2}\right) - \frac{n}{2} + 1\right) + (n-1) \\ &= n\lg\left(\frac{n}{2}\right) - n + 2 + n - 1 \\ &= n(\lg n - 1) + 1 \\ &= n\lg n - n + 1 \\ &= T(n) \\ &= LHS. \end{aligned}$$

///.

THE HIGHEST ORDER TERM IN $T(n)$ IS $n \lg n$, THUS

$$T(n) = \Theta(n \lg n).$$

WE'LL SEE IN CHAPTER 4 THAT THIS IS

TRUE EVEN WHEN n IS NOT A POWER OF 2.

ALL THIS IS SOMEWHAT VAGUE UNTIL WE DEFINE PRECISELY WHAT IS MEANT BY $\Theta(\cdot)$, BUT WE CAN SEE THAT FOR SUFFICIENTLY LARGE n

$$n \lg n << n^2 ,$$

SO THAT FOR LARGE n, MergeSort IS CONSIDERED MORE EFFICIENT THAN InsertionSort.

THIS IS TYPICAL OF THE WAY WE ANALYSE RECURSIVE ALGORITHMS. LET $T(n)$ BE THE RUNNING TIME (OR NUMBER OF 'BAROMETER' OPERATIONS) PERFORMED ON INPUTS OF 'SIZE' n. IF n IS SMALL, SAY $n \leq n_0$ FOR SOME CONSTANT n_0, NO SUBDIVISIONS ARE NECESSARY AND THE SOLUTION TAKES CONSTANT TIME:

$$T(n) = c = \text{const.}$$

THIS IS THE POINT AT WHICH THE RECURSION 'BOTTOMS OUT'.

IF $n > n_0$ WE DIVIDE THE PROBLEM INTO a SUBPROBLEMS, EACH OF SIZE $\frac{1}{b}$ TIMES THAT OF THE ORIGINAL (i.e. OF SIZE $\frac{n}{b}$).

SUPPOSE OUR ALGORITHM TAKES TIME $D(n)$ TO DIVIDE THE PROBLEM INTO SUBPROBLEMS, AND TIME $C(n)$ TO COMBINE SOLUTIONS TO THE SUBPROBLEMS INTO A SOLUTION TO THE ORIGINAL PROBLEM. WE OBTAIN THE RECURRENCE

$$T(n) = \begin{cases} c & n \le n_0 \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & n > n_0 \end{cases}$$

WE WILL LEARN IN CHAPTER 4 HOW TO SOLVE RECURRENCES OF THIS FORM, BOTH EXPLICITELY, AND IN THE ASYMPTOTIC SENSE.

READ: HANDOUT ON ASYMPTOTIC GROWTH RATES