CS340 - Analysis of Algorithms

Divide and Conquer

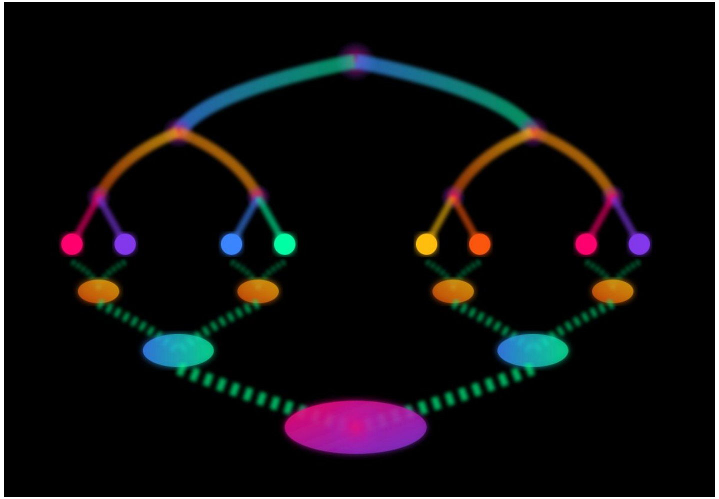
Announcements:

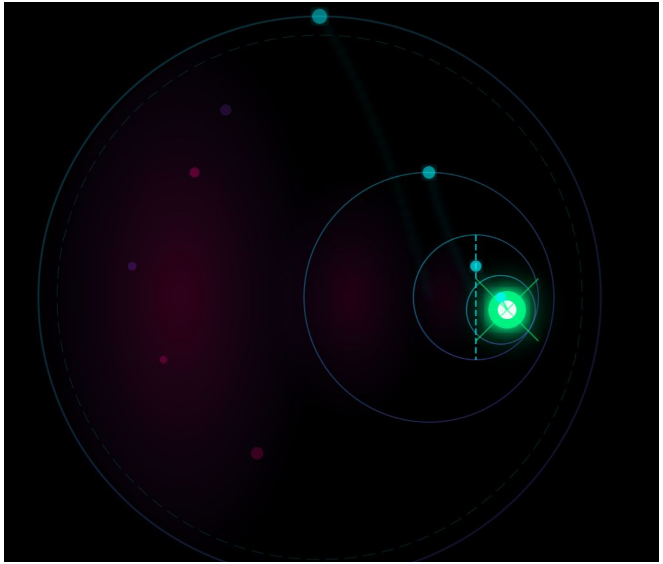
New OH times:

1. Monday 4-5pm

2. Friday 3-4pm

Divide and Conquer





Divide and Conquer

- A class of algorithms where you
  - *Divide:* break the input into several parts
  - *Conquer:* solve the problems to each part recursively
  - and *Combine*: the solutions to the subproblem into an overall solution

- Usually involves recursion

MergeSort

Divide and Conquer algorithm

1. **Divide**: recursively break down the problem into sub-problems
2. **Conquer:** recursively solve the sub-problems
3. **Combine:** combine the solutions to the sub-problems until they are a solution to the entire problem

Binary search is a divide and conquer algorithm

Usually involves recursion

Merge Sort

1. **Divide**: Divide the unsorted list into lists with only one element

2. **Conquer**: merge them back together in a sorted manner

3. **Combine:** merge the sorted sequences

Merge Sort

Sort a sequence of numbers A, $|A| = n$

Base: $|A| = 1$, then it's already sorted

General

- divide: split A into two halves, each of size $\frac{n}{2}$ ($\left\lfloor\frac{n}{2}\right\rfloor$ and $\left\lceil\frac{n}{2}\right\rceil$)
- conquer: sort each half (by calling mergeSort recursively)
- combine: merge the two sorted halves into a single sorted list

Example

6	8	4	1	7	2	5	3

Example

6	8	4	1	7	2	5	3

6	8	4	1

7	2	5	3

Example

6	8	4	1	7	2	5	3

6	8	4	1

7	2	5	3

6	8

4	1

7	2

5	3

Example

6 8 4 1 7 2 5 3

6 8 4 1   7 2 5 3

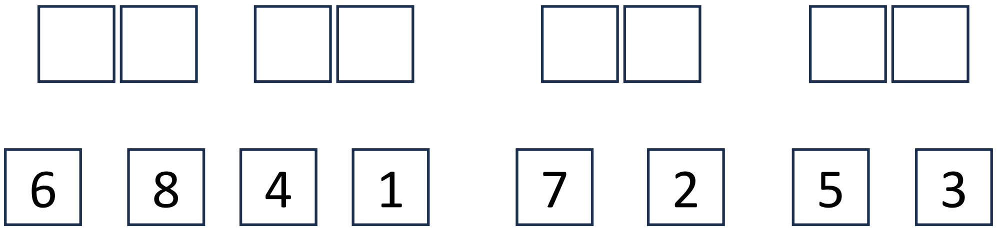
6 8   4 1   7 2   5 3

6   8   4   1   7   2   5   3

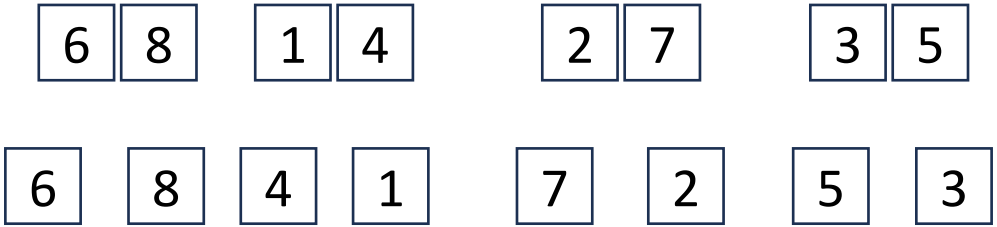
Example

6 8 4 1 7 2 5 3

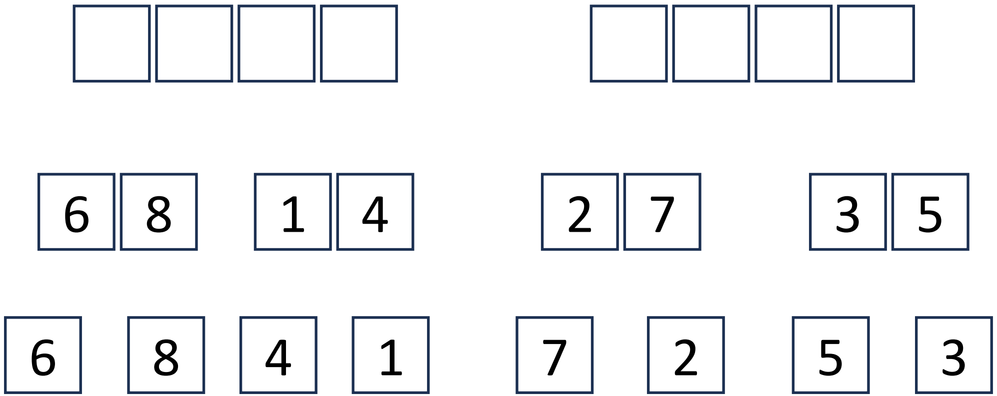
Example



Example



Example



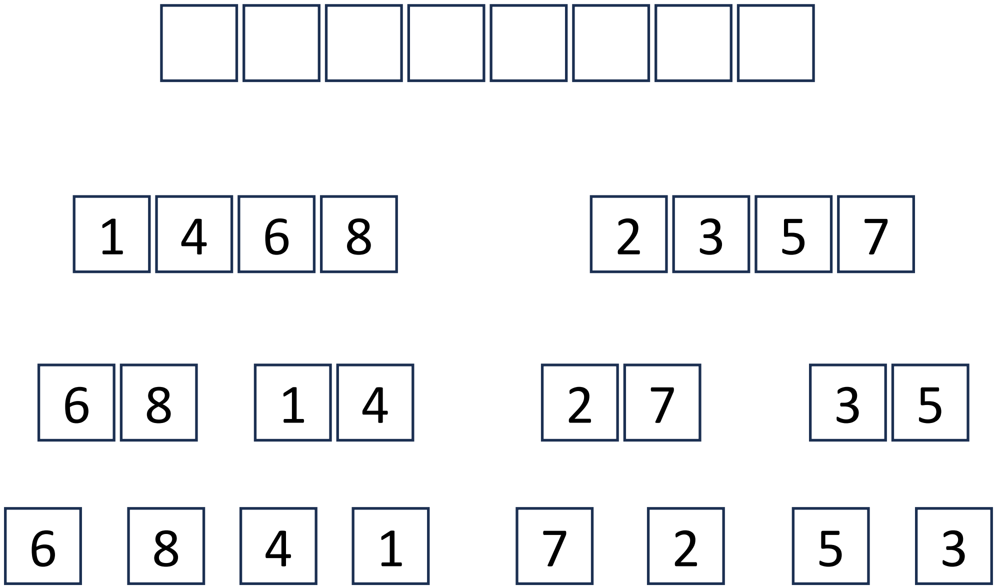
Example

1 4 6 8   2 3 5 7

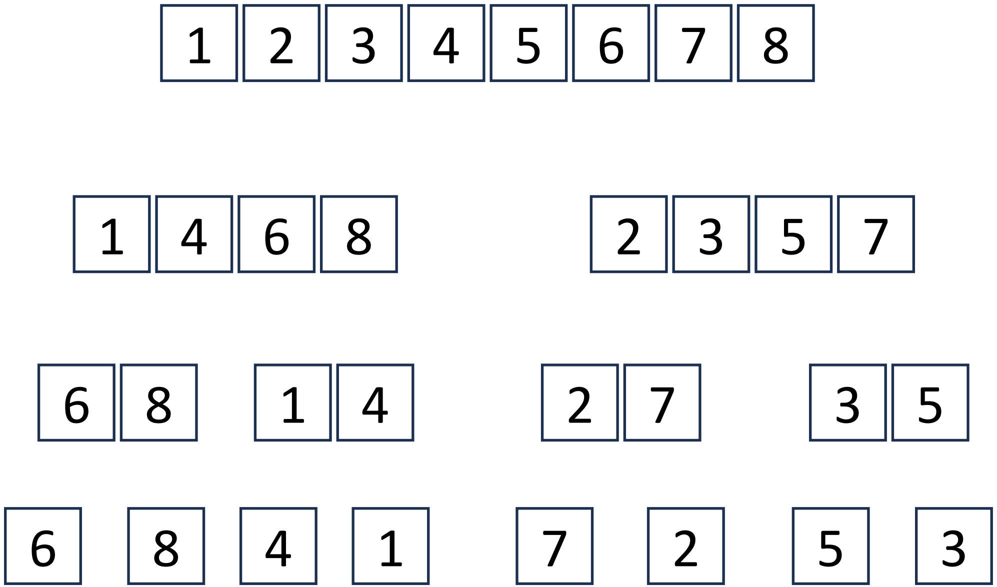
6 8   1 4   2 7   3 5

6   8   4   1   7   2   5   3

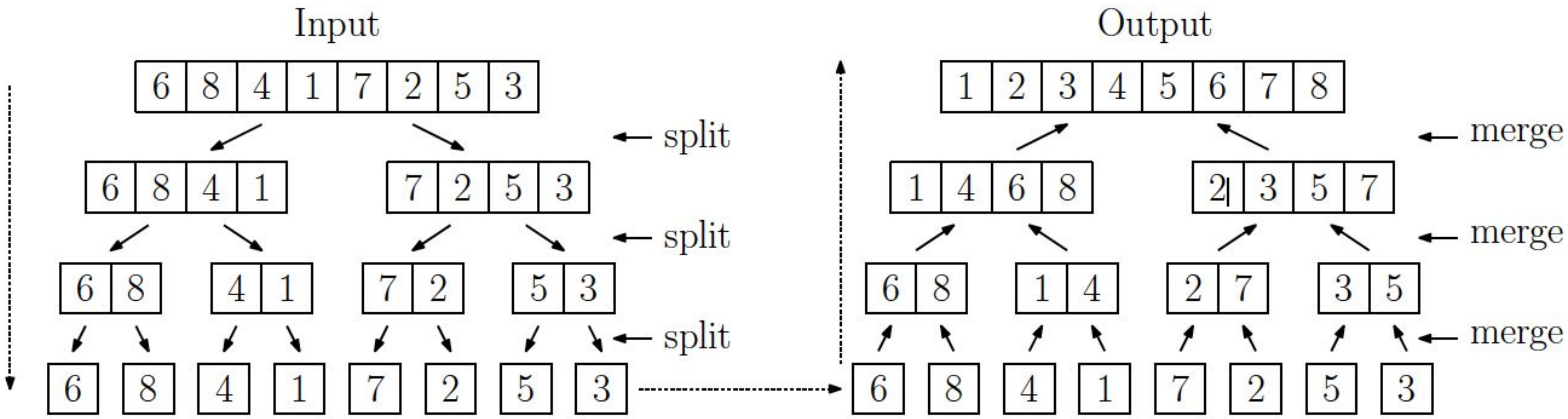
Example



Example



Example - summary

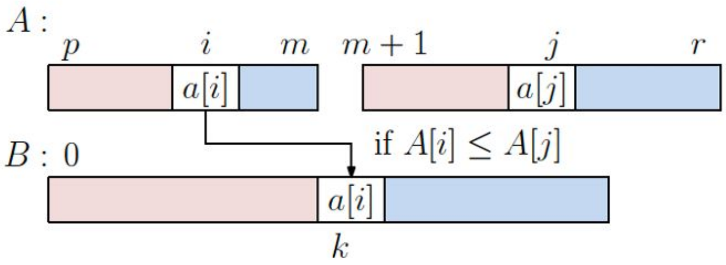


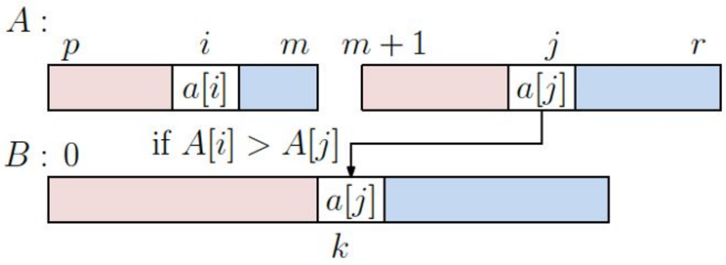
Merge Sort

https://youtu.be/4VqmGXwpLqc?si=WpYuXYLtJOuhvd77&t=24

The Merge

- Merge two sorted sub-arrays A[p:m] and A[m+1:r]





- use a temp array B
- maintain two indices i, j, k
  - i is index in subarray 1
  - j is index in subarray 2
  - k is index in B

Related by name only: https://blog.samaltman.com/the-merge

Merge - how do we sort two sorted lists?

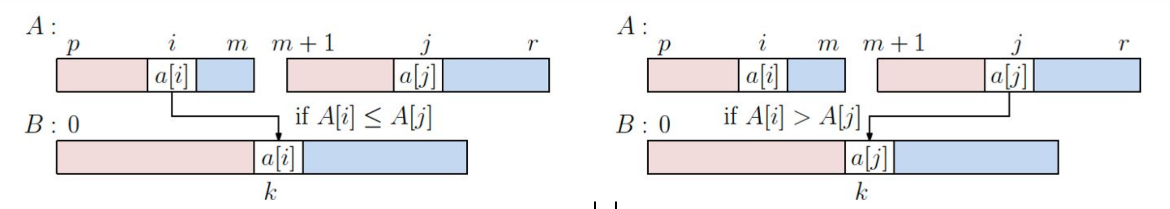
```
//combines 2 sorted sub-arrays A[p:m] and A[m+1:r]
Algorithm merge(A, p, m, r)
  B = [] //of size r-p+1
  i=p; j=m+1; k=0;

  while(i<m and j<=r)
    if A[i] < B[j]
      B[k++] = A[i++]
    else
      B[k++] = A[j++]


  while (i <= m)
      B[k++] = A[i++]
  while (j<=r)
      B[k++] = A[j++]
  Copy B back to A
```

runtime complexity?
O(n)

where n is r-p+1



Merge Sort Psuedo Code

```
mergeSort(A, p, r){
  if (p<r) {
    m = (p+r)/2
    mergeSort(A, p, m)
    mergeSort(A, m+1, r)
    merge(A, p, m, r)
  }
}
```

```
merge(A, p, m, r){
  new B[0, r-p]
  i=p; j=m+1; k=0
  while(i<=m and j<=r){
    if (A[i]<=A[j])
      B[k++] = A[i++]
    else
      B[k++] = A[j++]
  }
  while(i<=m) B[k++]=A[i++]
  while(j<=r) B[k++]=A[j++]
  copy B back to A
}
```

Runtime of MergeSort - Intuitive

Runtime of merging two sorted two lists A, B where $|A| + |B| = n$:

$O(n)$

How many times do we merge two sorted lists?

log n times

So total runtime is:

$O(n * \log(n))$

Analysis with Recurrence Relations

Abstract behavior of MergeSort (and many other Divide and Conquer algorithms)

- Divide the input into two pieces of equal size
- Solve the two subproblems on these pieces separately by recursion
- Combine the two results into an overall solution
- Spend only linear time for the initial division and final recombining

Base case: for MergeSort, when the input has been reduced to size 1

Analysis with Recurrence Relations

- + Divide the input into two pieces of equal size
- + Solve the two subproblems on these pieces separately by recursion
- + Combine the two results into an overall solution
- + Spend linear time for the initial division and final recombining

For any algorithm that fits this pattern, let T(n) denote its worst case runtime

Assuming n is even, and the algorithm spends O(n) time to divide the input into two pieces, and it spends T(n/2) to solve each one, and spends O(n) to combine the solution:

$$T(n) = \begin{cases} 1, n = 1 \\ 2T\left(\frac{n}{2}\right) + n, n > 1 \end{cases}$$

Unrolling the Recurrence Relation

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)\right) + n = 2^2T\left(\frac{n}{2^2}\right) + 2n$$

$$= 2\left(2\left(2T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)\right) + \left(\frac{n}{2}\right)\right) + n = 2^3T\left(\frac{n}{2^3}\right) + 3n$$

$$= \cdots$$

$$= 2^kT\left(\frac{n}{2^k}\right) + kn$$

Solving the Recurrence Relation

What is k here?

- The number of times we recursed

In mergesort, we stop when $n / 2^k = 1$

What is k equal to at that point?

- $k = \log n$

$T(n) = 2^{\log n} T(1) + n\log n = n + n\log n$

$= O(n\log n)$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

Master Theorem

- Formula for solving recurrence relations of the form: $T(n) = aT\left(\frac{n}{b}\right) + n^{d}$

- What is more expensive?
  - solving the subproblems? (recursive calls to mergesort)
  - dividing and conquering? (merging the sorted halves)

- Compares the cost of solving the subproblems, $n^{\log\_b(a)}$ with the cost of dividing and conquering n^d

Master Theorem

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

How to apply it:

1. Identify a, b, and d from the recurrence relation
2. Calculate $\log_b(a)$
3. Compare d to logb(a) to determine which of the three cases applies

Master Theorem Merge Sort

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

a = 2, b = 2, d = 1

$\log_2(2) = 1 = d$

CASE 2: O(nlogn)

Master Theorem Example 1

T (n) = 9T (n/3)+ n

a = 9, b = 3, d =1

$\log_3 (9) = 2 > d$

CASE 3: $O(n^2)$

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Master Theorem Example 2

T (n) = T (2n/3)+1

a = 1, b = 3/2, d = 0

$\log_{3/2}(1) =$

$= 0$

CASE 2: O(logn)

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Master Theorem Example 3

T (n) = 3T (n/4) + n log n

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

a = 3, b = 4, d = 1

$\log_4 (3) < d$?

Yes.

CASE 3: O(n) or O(nlogn)

Worksheet

https://bmc-cs-340.github.io/7-Recurrences.pdf

Summary

Merge Sort: A divide and conquer algorithm

O(nlogn)

Analyzed using unrolling OR masters theorem

Next class:

- Correctness of Merge Sort