DESIGN & ANALYSIS of ALGORITHMS

unit-2

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

Design and Analysis of Algorithms

BRUTE FORCE APPROACH

- When in doubt, use brute force - Ken Thompson
- Straightforward approach: directly based on problem statement (tends to naturally come to mind)
- Exhaustive search or generate and test
- Systematically enumerating all possible candidates for the solution
- Check whether each candidate satisfies the solution
- Inefficient approach

Q: Brute force algorithm to find divisors of a natural number

- enumerate all natural numbers less than n and check if the number divides n
- 1 to n must be checked; exhaustive

Q: 8 queens puzzle: in a 64 x 64 board, all 8 queens placed such that they do not clash (diff diagonals, rows, and diagonals).

- check all enumerations (possibilities) of the 8 queens
- $(64)^8$ possibilities ; very inefficient

Q: Brute force search: linear search

(grows linearly)

```
void search(int *a, int n, int key) {
    for (int i = 0; i < n; ++i) {
        if (a[i] == key) {
            printf("%d found at index %d\n", key, i);
            return;
        }
    }
    printf("%d not found\n", key);
}
```

for 100,000 elements, ~ 0.3 milliseconds

for 1,000,000 elements, ~ 2.5 milliseconds

BRUTE FORCE sorting algorithms

Selection sort

see unit 1, page 3

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-1-i\cancel{-1}\cancel{+1}) = \sum_{i=0}^{n-2} n-i-1$$

$$= (n-1) + (n-2) + \dots + 1$$

$$\boxed{C(n) = \frac{n(n-1)}{2} \in \Theta(n^2)}$$

bubble sort

- adjacent comparison sort

Algorithm

```
for i = 0 to i = n-2:
    for j = 0 to j = n-i-1:
        if arr[j+1] < arr[j]:
            swap arr[j+1] and arr[j]
```

Example : 5, 3, 2, 8, 3

pass = 1   3, 5, 2, 8, 3   j: 0 to 3

3, 2, 5, 8, 3

3, 2, 5, 3, 8

pass = 2   2, 3, 5, 3, 8   j: 0 to 2

2, 3, 3, 5, 8

pass = 3   2, 3, 3, 5, 8

pass = 4   2, 3, 3, 5, 8

pass = 5   2, 3, 3, 5, 8

efficient bubble sort

- if no swaps done in any iteration, the array is sorted and the program is terminated
- worst-case time unchanged; best-case time improved
- using a flag for sorted (assume to be sorted at every iteration; mark as unsorted once a swap is made)

Algorithm

```
for i = 0 to i = n-2:

    sorted = true
    for j = 0 to j = n-i-1:

        if arr[j+1] < arr[j] :
            swap arr[j+1] and arr[j]
            sorted = false

    if sorted is true:
        break
```

Time Complexity

$$C(n) = \sum_{i=0}^{n-2} \sum_{i=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} n-1-i$$

$$= (n-1) + (n-2) + \ldots + 1$$

$$\boxed{C(n) = \frac{n(n-1)}{2} \in \Theta(n^2)}$$

BRUTE FORCE *searching algorithms*

- search for a pattern in a given text

Sequential search

page 3 - linear search

String matching

eg: pattern: "TEXT" (length = m), text: below (length = n)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	R	E	A	D		T	E	X	T	B	O	O	K	S
$i = 0$	T	E	X	T										
$i = 1$		T	E	X	T									
$i = 2$			T	E	X	T								
$i = 3$				T	E	X	T							
$i = 4$					T	E	X	T						
$i = 5$						T	E	X	T					

align and slide until match found

- worst case = text length - pattern length + 1 iterations
- 0 to n-m or n-m+1 trials

String matching

```c
int find(char *text, char *pattern) {
    int n = strlen(text);
    int m = strlen(pattern);
    int i, j;

    for (i = 0; i <= n - m; ++i) {
        j = 0;
        while ((j < m) && (pattern[j] == text[i+j])) {
            ++j;
        }

        if (j == m) {
            break;
        }
    }

    if (i > n-m) {
        return 0;
    }
    return 1;
}
```

Output

```
➜  1-4 String Matching ./find
Enter text:
textbook
Enter pattern:
book
1
➜  1-4 String Matching ./find
Enter text:
textbook
Enter pattern:
boy
0
```

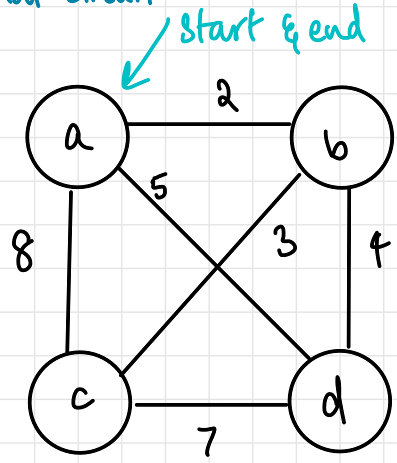
EXHAUSTIVE SEARCH

- generate a set of all potential solutions systematically
- compute cost of each permutation
- find optimal solution by comparing every solution with every other solution

travelling salesperson

- a salesperson has to travel from source to dest with minimum tour cost (weights/distances)
- brute force solution - find every possible permutation of n-1 cities $[1, \underbrace{\ldots\ldots}_{\text{all possibilities}}, n-1]$
- exponential growth - NP problem $(n-1)!$
- Hamiltonial Circuit - visit all vertices in a graph exactly once, start and end on the same vertex
- Find shortest Hamiltonian circuit in a weighted connected graph

Q: Find optimal circuit



$3! = 6$ permutations

$a \to b \to c \to d \to a = 2+3+7+5 = 17$

$a \to b \to d \to c \to a = 2+4+7+8 = 21$

$a \to c \to b \to d \to a = 8+3+4+5 = 20$

$a \to c \to d \to b \to a = 8+7+4+2 = 21$

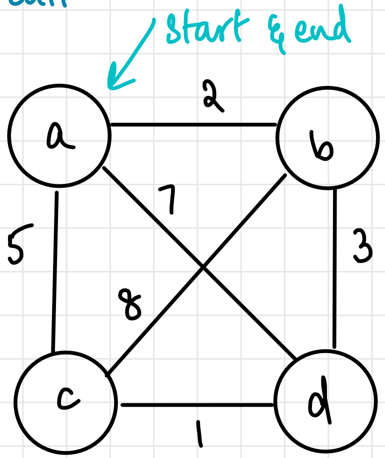
$a \to d \to b \to c \to a = 5+4+3+8 = 20$

$a \to d \to c \to b \to a = 5+7+3+2 = 17$

Optimal paths:

$a \to b \to c \to d \to a$
$a \to d \to c \to b \to a$

Q: Find circuit



$a \to b \to c \to d \to a = 2+8+1+7 = 18$

$a \to b \to d \to c \to a = 2+3+1+5 = 11$

$a \to c \to b \to d \to a = 5+8+3+7 = 23$

$a \to c \to d \to b \to a = 5+1+3+2 = 11$

$a \to d \to b \to c \to a = 7+3+8+5 = 23$

$a \to d \to c \to b \to a = 7+1+8+2 = 18$

Optimal solution

$a \to b \to d \to c \to a$

$a \to c \to d \to b \to a$

KNAPSACK PROBLEM

- bag with capacity M, objects with weights and values
- 0/1 knapsack; object either picked up or not (no fractions)
- Optimisation: maximise profit due to objects
- eg: a thief tries to maximise profit with finite bag size (weight and value)
- exhaustive search (all subsets found, value and weight calculated, optimised subset found)
- 2^n subsets

Q: Knapsack capacity W=16

Item	Weight	Value
1	2	20
2	5	30
3	10	50
4	5	10

	Subset	Weight	Value
1	{}	0	0
2	{1}	2	20
3	{2}	5	30
4	{3}	10	50
5	{4}	5	10

6	{1,2}	7	50
7	{1,3}	12	70
8	{1,4}	7	30
9	{2,3}	15	80 optimal
10	{2,4}	10	40
11	{3,4}	15	60
12	{1,2,3}	17	not feasible
13	{1,2,4}	12	60
14	{1,3,4}	17	not feasible
15	{2,3,4}	20	not feasible
16	{1,2,3,4}	22	not feasible

- Exhaustive search: $\Omega(2^n)$

Job Assignment PROBLEM

- There are n people who need to be assigned to n jobs
- The cost of assigning a job j to a person i is $C[i,j]$
- Minimisation problem

	J_1	J_2	. . . . .	J_n
P_1	c_{11}	. . .		
⋮	.	. .	.	
P_n	.			

Q:

	J1	J2	J3	J4
P1	9	2	7	8
P2	6	4	3	7
P3	5	8	1	8
P4	7	6	9	4

n! permutations

fit n people into n jobs (n!)

4! = 24 possibilities

J1-J2-J3-J4

1-2-3-4
1-2-4-3
1-3-2-4
1-3-4-2
1-4-2-3
1-4-3-2

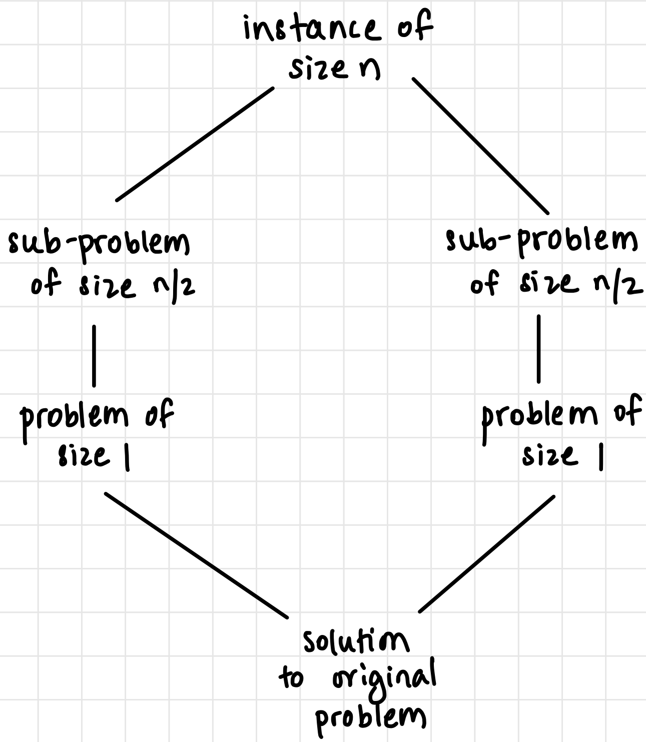
2-1-3-4
2-1-4-3
2-3-1-4
2-3-4-1
2-4-1-3
2-4-3-1

3-1-2-4
3-1-4-2
3-2-1-4
3-2-4-1
3-4-1-2
3-4-2-1

4-1-2-3
4-1-3-2
4-2-1-3
4-2-3-1
4-3-1-2
4-3-2-1

DIVIDE & CONQUER

- big problem divided into smaller sub-problems of same nature
- solve smaller problems recursively
- combine the solutions



$$\boxed{T(n) = a * T(n/b) + f(n)}$$

time spent on dividing and combining

b instances

Master's Theorem

Pg 35, unit 1

for the recurrence

$$T(n) = a^{*}\, T(n/b) + f(n)$$

if $f(n) \in \Theta(n^{d})$ where $d \geq 0$

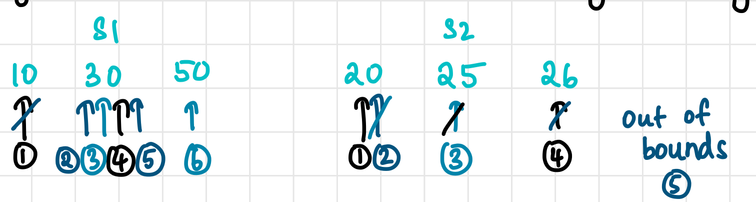
if $a < b^{d}$, $T(n) \in \Theta(n^{d})$

if $a = b^{d}$, $T(n) \in \Theta(n^{d} \log n)$

if $a > b^{d}$, $T(n) \in \Theta(n^{\log_b a})$

merge sort

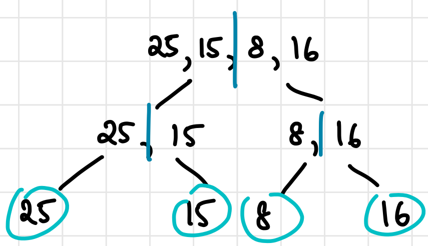
- Merge: merge 2 sorted sets to form a larger merged set



10, 20, 25, 26, 30, 50

set exhausted

- Divide: Keep dividing into halves until size of array = 1 (each element is a sorted array of size 1)



→ elementary case (size = 1)

```
Algorithm MergeSort(A[0...n-1])
  // recursive
  // input: array A[0...n-1]
  // output: sorted array A[0....n-1]

  if n > 0
      copy A[0... ⌊n/2⌋ - 1] to B[0 ... ⌊n/2⌋ - 1]
      copy A[⌊n/2⌋ ... n-1] to C[0 .... ⌊n/2⌋ - 1]
      MergeSort(B[0....⌊n/2⌋ - 1)
      MergeSort(C[0...⌊n/2⌋ - 1)
      Merge(B, C, A)

Algorithm Merge(B[0...p-1], C[0...q-1], A[0 ... p+q-1])
  // Merge two sorted arrays into one sorted array
  // input: arrays B[0...p-1] and C[0...q-1], both sorted
  // output: array A[0... p+q-1] of both arrays' elements, sorted

  i=0, j=0, k=0

  while i<p and j<q
      if (B[i] ≤ C[j])
          A[k] = B[i]
          i = i+1
      else
          A[k] = C[j]
          j = j+1
      k = k+1

  if i=p
      copy C[j... q-1] to A[k... p+q-1]

  else
      copy B[i... p-1] to A[k... p+q-1]
```

Q: Show merge sort of 8, 3, 2, 9, 7, 1, 5, 4

low = 0
high = 7
mid = (0+7)/2
= 3

0	1	2	3	4	5	6	7
8	3	2	9	7	1	5	4

8, 3, 2, 9   7, 1, 5, 4

8, 3   2, 9   7, 1   5, 4

8   3   2   9   7   1   5   4

3, 8   2, 9   1, 7   4, 5

2, 3, 8, 9   1, 4, 5, 7

1, 2, 3, 4, 5, 7, 8, 9

$T(n) = 2T(n/2) + n-1$ ← merge time (comparisons)

S_1 x_1 x_2 x_3

1   2   3   4   5

S_2 x_4 x_5 x_6

5 comparisons

Master's Theorem: $a = 2$, $d = 1$, $b = 2$

$a = b^d$ $(2 = 2)$

$\therefore \boxed{T(n) \in \Theta(n \log n)}$ ← improvement over $\Theta(n^2)$

implementation in C

```c
void merge_sort(int *A, int l, int h) {
    if (l < h) {
        int m = (l + h)/2;

        merge_sort(A, l, m);
        merge_sort(A, m+1, h);
        merge(A, l, m, h);
    }
}

void merge(int *A, int l, int m, int h) {
    int B[MAX];

    int i = l, j = m + 1, k = 0;
    while (i <= m && j <= h) {
        if (A[i] <= A[j]) {
            B[k++] = A[i++];
        }
        else {
            B[k++] = A[j++];
        }
    }
    while (i <= m) {
        B[k++] = A[i++];
    }
    while (j <= h) {
        B[k++] = A[j++];
    }

    k = 0;
    for (i = l; i <= h; ++i, ++k) {
        A[i] = B[k];
    }
}
```

for 1,000 elements
~0.2 ms

for 10,000 elements
~ 1.7 ms

for 100,000 elements
~ 20 ms

for 1,000,000 elements
~ 200 ms

quick sort

5 3 1 9 10 4 6 28

- Hoare's partition method
- pivot element
- variations of QS
- pivot: first element in array
- divide into 2 groups: A[i] < p and A[i] > p ← pivot

5	3	1	9	10	4	6	28	n=8
p	i						j	

i = low + 1
j = high
p = pivot

5	3	1	9	10	4	6	28
p	i	i	i		j	j	j
	3<5	1<5	9>5		4<5	6>5	28>5
	✓	✓	stop		stop	✓	✓

swap

5	3	1	4	10	9	6	28
			i	i j	j		
			j	10>5			
				stop			
			4<5	10>5			
			stop	✓			

swap j^{th} and pivot when i and j cross p i j

$\underbrace{4 \quad 3 \quad 1}_{\text{LHS}}$ 5 $\underbrace{10 \quad 9 \quad 6 \quad 28}_{\text{RHS}}$

partitioning successful

pivot

quick sort from low to j-1

quick sort from j+1 to high

1 3 4

$\underbrace{6 \quad 9}_{\text{qsort}}$ 10 28

Q: Show qsort for the array

5 28 7 14 10 3 2 1

p i j

i stop $28 \not< 5$

j stop $1 \not> 5$

$i < j \rightarrow$ swap i & j

5 1 7 14 10 3 2 28

i stop $7 \not< 5$

j stop $2 \not> 5$

$i < j \rightarrow$ swap i & j

5 1 2 14 10 3 7 28

i stop $14 \not< 5$

j stop $3 \not> 5$

$i < j \rightarrow$ swap

5 1 2 3 10 14 7 28

j ij

j stop i stop

j < i → swap p and j

3 1 2 5 10 14 7 28

3 1 2

p i ji i

j stop stop

swap j & p

2 1 3

p ij i → OOB

j stop stop

swap j & p

1 2 3

10 14 7 28

p i j j

stop stop 28 > 10 ✓

swap i & j

10 7 14 28

j i

stop stop

swap p & j

7 10 14 28

14 28

p ij

Final array

1 2 3 7 10 14 28

```
Algorithm QuickSort (A[l...h])

  // Sorts a subarray by quicksort
  // Input: a subarray A[l...h] of A[0...n-1]
  // Output: a subarray A[l...h] sorted in ascending order

  if l < h

     s = Partition(A[l...h])
     QuickSort(A[l...s-1])
     QuickSort(A[s+1...h])
```

```
Algorithm Partition (A[l...h])
  // Partitions a subarray using first element as pivot
  // Input: a subarray A[l...h] of A[0...n-1]
  // Output: a partition of A[l...h], with the split position returned

  p = A[l]
  i = l
  j = h

  repeat
     repeat i = i+1 until A[i] ≥ p
     repeat j = j-1 until A[j] ≤ p
     swap(A[i], A[j])
  until i ≥ j

  if j ≠ l
     swap (A[l], A[j])

  return j
```

IMPLEMENTATION IN C

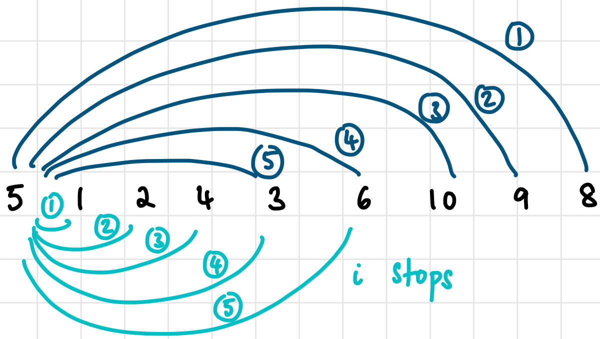
```c
void quicksort(int *a, int low, int high) {
    int j;
    if (low < high) {
        j = partition(a, low, high);
        quicksort(a, low, j - 1);
        quicksort(a, j + 1, high);
    }
}

int partition(int *a, int low, int high) {
    int pivot = a[low];
    int i = low + 1;
    int j = high;

    while (i <= j) {
        while (i <= high && a[i] <= pivot) {
            ++i;
        }
        while (j > low && a[j] >= pivot) {
            --j;
        }
        if (i < j) {
            // Swap
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    // Crossover
    if (j != low) {
        a[low] = a[j];
        a[j] = pivot;
    }
    return j;
}
```

can also be strict (repetitions)

efficiency class

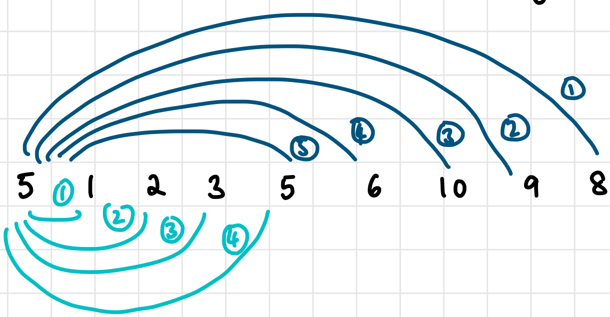
best case

- Best case: split 50-50



$n = 9$

comparisons = $n+1$ (no repeating of pivot)



$n = 9$

comparisons = n

basic op

$$C_{best}(n) = 2\,C_{best}(n/2) + n$$

$a = 2$

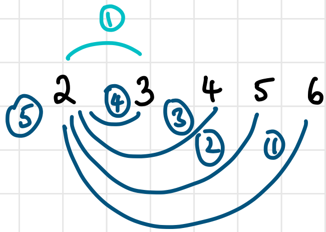
$b = 2$

$d = 1$

$a = b^d$

$$T(n) = \Theta(n \log n)$$

worst CASE



comparisons = $n+1$

$$C_{worst} = (n+1) + n + \ldots + 3 = \frac{(n+1)(n+2)}{2} - 3$$

$$T(n) = \Theta(n^2)$$

average CASE

- partitioning can take place at n different locations

$$C_{avg}(n) = \frac{1}{n} \sum_{s=0}^{n-1} [(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n>1$$

$$C_{avg}(n) \approx 2n \ln(n) \approx 1.38\, n \log_2 n$$

BINARY SEARCH

- decrease and conquer — size decreases (only 1 subinstance)
- two conditions
  1. the elements must be in order (sorted array)
  2. every element randomly accessible (not linked list)

example

key = 21

8	21	32	65	72	89	100
0	1	2	3	4	5	6
low						high

$$\text{mid} = \frac{\text{low}+\text{high}}{2} = 3$$

key < array[mid]

binary search on subarray

8	21	32	65	72	89	100
0	1	2	3	4	5	6
low						high

```
Algorithm BinarySearch( A[low...high], k)
  // Implements recursive binary search
  // Input: array A[0...n-1] sorted in ascending order
  // Output: index of element k, if found, or -1 otherwise

  if low ≤ high
      mid = (high+low)/2
      if A[mid] == k
          return k
      else if A[mid] > k
          return BinarySearch(A[low.... mid-1], k)
      else
          return BinarySearch(A[mid+1 ... n-1], k)

  else
      return -1
```

```
Algorithm BinarySearch( A[0...n-1], k)
  // Implements non-recursive binary search
  // Input: array A[0...n-1] sorted in ascending order
  // Output: index of element k, if found, or -1 otherwise

  while low ≤ high
      mid = (high+low)/2
      if A[mid] == k
          return k
      else if A[mid] > k
          high = mid-1
      else
          low = mid+1
  return -1
```

TIME COMPLEXITY

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

$a = 1$
$b = 2$
$d = 0$

$a = b^d$

$$\boxed{T(n) = \Theta(\log n)}$$

unsorted set

$$T(n) = \max(T_{sort}(n), T_{BS}(n))$$

$$T(n) = n \log n$$

IMPLEMENTATION IN C

iterative

```c
int bs_iter(int *a, int n, int key) {
    int low = 0, high = n - 1, mid;

    while (low <= high) {
        mid = (low + high)/2;

        if (a[mid] == key) {
            return mid;
        }
        else if (a[mid] > key) {
            high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }
    return -1;
}
```

recursive

```c
int bs_recur(int *a, int low, int high, int key) {
    int mid;

    if (low <= high) {
        mid = (low + high)/2;

        if (a[mid] == key) {
            return mid;
        }
        else if (a[mid] > key) {
            return bs_recur(a, low, mid - 1, key);
        }
        else {
            return bs_recur(a, mid + 1, high, key);
        }
    }
    return -1;
}
```

binary trees

- non-linear data structure
- finite set of nodes
- either empty or 3 subsets (root, left subtree, right subtree)
- faster insertions and deletions
- traversals: refer Data Structures, sem 3

disjoint sets (both binary trees)

height of a tree

height of empty = -1

height of tree = 1+ max(height(left), height(right))

Algorithm Height (T)

if $T = \phi$ // empty tree
    return -1

return 1+ max (Height(T_L), Height(T_R))

Recurrence Relation

basic operation: addition

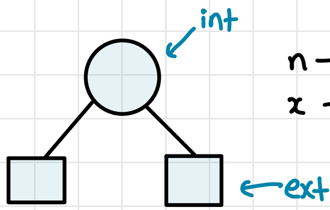
$$A(n) = A(n_{T_L}) + A(n_{T_R}) + 1$$

non-symmetric; not straightforward

basic operation: comparison $(T = \phi)$

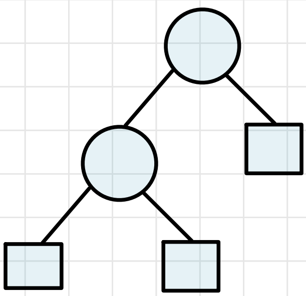
external node: empty set
internal node: non-empty set

1) one internal node



$n \rightarrow$ int $= 1$
$x \rightarrow$ ext $= 2$

2) two internal nodes



$n = 2$
$x = 3$

$\boxed{x = n+1}$

total nodes $= x + n$

$$\begin{aligned} \text{total comparisons} &= x + n \\ &= n + 1 + n \\ &= 2n + 1 \\ &= \Theta(n) \end{aligned}$$

number of additions $= n$

MULTIPLICATION OF LARGE INTEGERS

$$
\begin{array}{r}
\overset{1}{4}\ 2 \\
\times\ 3\ 5 \\
\hline
2\ 0\ 0 \\
1\ 2\ 6\ 0 \\
\hline
1\ 4\ 6\ 0
\end{array}
$$

total: 4 multiplications
(basic operation)

$a \times b$

divide digits (even)

$a = a_1 a_0$

$b = b_1 b_0$

$a = a_1 \times 10^{n/2} + a_0$

$b = b_1 \times 10^{n/2} + b_0$

pad with zeroes if odd

$$a \times b = (a_1 \times 10^{n/2} + a_0)(b_1 \times 10^{n/2} + b_0)$$

$$= (a_1 \times b_1) \times 10^n + (a_1 \times b_0 + a_0 \times b_1) \times 10^{n/2} + a_0 \times b_0$$

$$= c_2 \times 10^n + c_1 \times 10^{n/2} + c_0$$

① $c_0 = a_0 \times b_0$

② $c_2 = a_1 \times b_1$

rewrite to reduce # of multiplications

③ $c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$

Q: $\underset{a_1}{3}\underset{a_0}{3} \times \underset{b_1}{2}\underset{b_0}{4}$ $n=2$

$c_2 = a_1 \times b_1 = 6$

$c_0 = a_0 \times b_0 = 12$

$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$
$= 6 \times 6 - 18$
$= 18$

$33 \times 24 = 6 \times 10^2 + 18 \times 10 + 12$
$= 600 + 180 + 12$
$= 792$

Q: $\underset{a_1}{12}\underset{a_0}{33} \times \underset{b_1}{11}\underset{b_0}{24}$

$c_2 = 12 \times 11 \longrightarrow (1)$
$c_0 = 33 \times 24 \longrightarrow (2)$

$c_1 = (12+33) \times (11+24) - (12 \times 11 + 33 \times 24)$
$= (45 \times 35) - (12 \times 11 + 33 \times 24)$
$\longrightarrow (3)$

(1) $\underset{a_1}{1}\underset{a_0}{2} \times \underset{b_1}{1}\underset{b_0}{1}$

$c_2 = 1 \times 1 = 1 \longrightarrow (4)$
$c_0 = 2 \times 1 = 2 \longrightarrow (5)$
$c_1 = 3 \times 2 - (1+2)$
$= 6 - 3 = 3 \longrightarrow (6)$
$12 \times 11 = 1 \times 10^2 + 3 \times 10 + 2 = 132$

(2) 33 × 24
$a_1\ a_0 \quad b_1\ b_0$

$c_2 = 3 \times 2 = 6 \longrightarrow (7)$
$c_0 = 3 \times 4 = 12 \longrightarrow (8)$
$c_1 = 6 \times 6 - (12+6)$
$= 18 \longrightarrow (9)$

$33 \times 24 = 6 \times 10^2 + 18 \times 10 + 12$
$= 792$

(3) 45 × 35
$a_1\ a_0 \quad b_1\ b_0$

$c_2 = 4 \times 3 = 12 \longrightarrow (10)$
$c_0 = 5 \times 5 = 25 \longrightarrow (11)$
$c_1 = 9 \times 8 - (12+25)$
$= 72 - 37 \longrightarrow (12)$
$= 35$

$45 \times 35 = 12 \times 10^2 + 35 \times 10 + 25 = 1200 + 350 + 25 = 1575$

$c_2 = 12 \times 11 = 132$
$c_0 = 33 \times 24 = 792$

$c_1 = 1575 - (132 + 792)$
$= 651$

$1233 \times 1124 = 132 \times 10^4 + 641 \times 10^2 + 792$
$= 1320000 + 64100 + 792$
$= 1385892$

$M(n) = 3M(n/2)$

$n = 2^k$

$k = \log_2 n$

$M(1) = 1$

$$\begin{aligned} M(2^k) &= 3M(2^{k-1}) \\ &= 3 \cdot 3\, M(2^{k-2}) \\ &= 3 \cdot 3 \cdot 3\, M(2^{k-3}) \\ &= 3^i\, M(2^{k-i}) \end{aligned}$$

$i = k$ $\quad = 3^k M(1)$

$= 3^k$

$$M(n) \in \Theta(3^{\log_2 n}) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585})$$

$A(n) = 3A(n/2) + cn$ for $n > 1$, $A(1) = 1$

$A(n) \in \Theta(n^{1.585})$

Strassen's MATRIX MULTIPLICATION

- For two 2×2 matrices, seven multiplications instead of 8 (brute force) $(2+2+2+2)$

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$ 8 additions

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$
$$m_2 = (a_{10} + a_{11}) * b_{00}$$
$$m_3 = a_{00} * (b_{01} - b_{11})$$
$$m_4 = a_{11} * (b_{10} - b_{00})$$
$$m_5 = (a_{00} + a_{01}) * b_{11}$$
$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$$
$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$$

- 10 additions + 8 additions = 18 additions

FOR ANY $n \times n$ MATRIX

- pad zeroes if not square
- divide matrices into submatrices

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$M(n) = 7\,M(n/2) \qquad M(1) = 1$

$n = 2^k \Rightarrow k = \log_2 n$

$$\begin{aligned} M(2^k) &= 7\,M(2^{k-1}) \\ &= 7 \cdot 7\,M(2^{k-2}) \\ &= 7^i\,M(2^{k-i}) \\ &= 7^k = 7^{\log_2 n} = n^{\log_2 7} = n^{2.807} < n^3 \end{aligned}$$

theory: can get n^2, not yet found

$T(n) \in \Theta(n^{2.807})$

Q: $\begin{bmatrix} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{bmatrix}$

$\quad A \qquad\qquad\qquad B$

Show initial steps. (not all)

$$m_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$$

$$= \left(\begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} \right) * \left(\begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix} \right)$$

$$= \begin{array}{cc} 00 & 01 \\ \multicolumn{2}{c}{\begin{bmatrix} 4 & 0 \\ 6 & 2 \end{bmatrix}} \\ 10 & 11 \end{array} \times \begin{array}{cc} 00 & 01 \\ \multicolumn{2}{c}{\begin{bmatrix} 1 & 2 \\ 7 & 1 \end{bmatrix}} \\ 10 & 11 \end{array} = \begin{bmatrix} (m_1 + m_4 - m_5 + m_7) & (m_3 + m_5) \\ (m_2 + m_4) & (m_1 + m_3 - m_2 + m_6) \end{bmatrix}$$

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$
$$= 6 \times 2 = 12$$

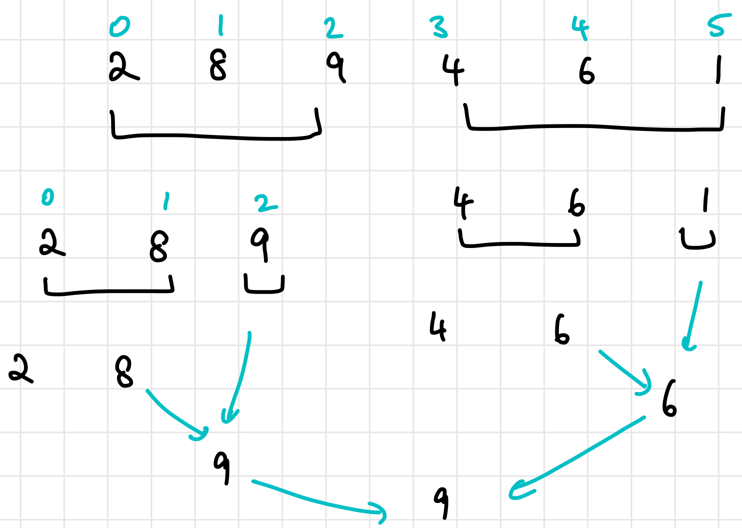
$$m_4 = a_{11} * (b_{10} - b_{00})$$
$$= 2 \times (7 - 1) = 12$$

$\vdots$

and so on

Q: Design an algorithm to find the max of elements using divide and conquer strategy.

```
Algorithm FindMax( A[0 ... n-1])
   // input: array of n elements
   // output : max element

   mid = (0+n-1)/2

   if size(A)=1
        return A[0]

   left_max = FindMax(A[0 ... mid])          ] low &
   right_max = Find Max(A[mid+1 ... n-1])    ]  high

   if left_max > right_max
        return left_max

   else
        return right_max
```



Code in C

```
int maxelement(int *a, int low, int high) {
    int mid = (low + high) / 2;

    if (low < high) {
        int max1 = maxelement(a, low, mid);
        int max2 = maxelement(a, mid+1, high);
        if (max1 > max2) {
            return max1;
        }
        else {
            return max2;
        }
    }
    return a[low];
}
```

```
→  2-5 Max Element ./max
Enter the number of elements: 7
Enter elements: 0 2 3 8 1 9 4
Max element: 9
```

Q: Find key element in a set of elements

like binary search, but search both halves

Q: Find a^n using brute force and divide and conquer

brute force: $a * a * a \ldots a$ n times

divide and conquer :
$$\begin{cases} a^{\lfloor n/2 \rfloor} * a^{\lceil n/2 \rceil} & , n > 1 \\ a & , n = 1 \end{cases}$$

Q: $T(n) = 4T(n/2) + n$, $T(1) = 1$

$a = 4$
$b = 2$
$d = 1$

$a > b^d$
$4 > 2$

$T(n) \in \Theta(n^{\log_b a})$

$T(n) \in \Theta(n^2)$

Q: $T(n) = 4T(n/2) + n^2$, $T(1) = 1$

$a = 4 \quad b = 2 \quad d = 2$

$a = b^d$
$4 = 4$

$T(n) \in \Theta(n^d \log n)$

$T(n) \in \Theta(n^2 \log n)$

Q: $T(n) = 4T(n/2) + n^3$, $T(1) = 1$

$a = 4 \quad b = 2 \quad d = 3$

$a < b^d$

$T(n) \in \Theta(n^d)$

$T(n) \in \Theta(n^3)$

Q: Given n positive integers, partition them into 2 disjoint subsets with the same sum of their elements (Partition Problem)

eg: 1 2 3 4 5 6

2, 4 & 6
1, 3, 4 & 3, 5

brute force: exhaustive search of all subsets and find subsets with the same sum

$T(n) \in \Theta(2^n)$

Q: Dutch flag problem: red, white, blue balls to be rearranged into order of red, white and then blue

eg: input: 0 1 2 0 1 2

output: 0 0 1 1 2 2

Algorithm #1: traverse through array once and count number of 0's, 1's and 2's and then replace array with sorted no.s

Algorithm #2: 3-way partition

000...	111...	unknown	222...

low   high
start
mid

a[mid] = 0 ⟶ swap with low and increment mid & low

a[mid] = 1 ⟶ correct partition
move mid right

a[mid] = 2 ⟶ swap with end and decrement end

https://www.geeksforgeeks.org/sort-an-array-of-0s-1s-and-2s/

Q: Quick sort using Dutch flag algorithm (3-way sort)

2 6 5 2 6 8 6 1 2 6 ← pivot

pivot = 6

normal ___QS___ 6 ___QS___

Dutch ___QS___ 6 6 ___QS___

Q: Index inversion problem

Output: count of II

if a[j] = i
and a[i] = j

$\Theta(n^2)$: run 2 loops like selsort and check for condition

0	1	2	3
1	0	3	2

$\Theta(n \log n)$: hint – use merge sort

termination condition:

0	1
1	0

0 1 2 3 4 5 6 7

1, 0, 3, 2, 4, 6, 5, 8

1, 0   3, 2   4, 6   5, 8

				4	5	6	7
1	0	3	2	4	6	5	8
i	j	i	j	i	j	i	j

if a[i] = j
a[j] = i
# inv ↑

if a[i] = j
a[j] = i
# inv ↑

if a[i] = j
a[j] = i
# inv ↑

if a[i] = j
a[j] = i
# inv ↑

merge normally

0	1	2	3	4	5	6	7
1,	0	3,	2	4,	6	5,	8
i		j		i	i	j	

if a[i] = j
a[j] = i
# inv ↑
inc both

else if a[i] < j
inc i