Sorting & Searching

Binary Search

Example of searching a name in the dictionary

In binary search algo, during each stage of our algorithm, our search for an ITEM is reduced to a segment of elements of DATA.

DATA[BEG], DATA[BEG+1], ..., DATA[END]

BEG & END are beginning & end locations of segment

→ The algo. compares ITEM with the middle element DATA[MID] of the segment. MID is obtained by

MID = INT((BEG+END)/2)

→ If DATA[MID] = ITEM search successful

→ If ITEM < DATA[MID]

Set END := MID-1 & begin searching

Else

BEG = MID+1 & Begin searching.

Initially BEG=1 & END=N or BEG=LB & END=UB

Algorithm: BINARY(DATA, LB, UB, ITEM, LOC)

DATA → Sorted array with lower bound LB & Upper Bound UB.

LOC → location of ITEM found

initially LOC = NULL

STEP 1 [Initialisation]
Set BEG := LB, END := UB & MID = INT((L
2. Repeat steps 3 & 4 while BEG ≤ END & DATA[
3. If ITEM < DATA[MID] then
Set END := MID - 1
Else
Set BEG := MID + 1
[End of If structure]
4. Set MID := INT ((BEG + END)/2)
[End of Step 2 loop]
5. If DATA[MID] = ITEM then
Set LOC := MID
Else
Set LOC := NULL
[End of If structure]
6. Exit

Example :
Given

11, 22, 30, 33, 40, 44, 55, 60, 66, 77,

Search ITEM = 40

1. Initially BEG = 1 & END = 13
MID = (1+13)/2 = 7 so DATA[MID] = 55
2. 40 < 55 END = MID - 1 = 6
MID = (1+6)/2 = 3 DATA[MID] = 30

3. Since $40 > 30$, BEG = MID + 1 = 4
MID = (4+6)/2 = 5 so DATA[MID] = 5
Hence LOC = MID = 5 Found.

(b) Search ITEM = 85

1. Again initially BEG = 1, END = 13, MID = 7
DATA[MID] = 55
2. $85 > 55$ BEG = MID + 1 = 8, END = 13
MID = (8+13)/2 = 21/2 = 10 DATA[MID] = 77
3. $85 > 77$ BEG = MID + 1 = 11
MID = (11+13)/2 = 12 so DATA[MID] = 88
4. $85 < 88$ END = MID - 1 = 11
MID = (11+11)/2 = 11 DATA[MID] = 80
As BEG = END = MID
ITEM not found

Sorting

INSERTION SORT

Suppose an array A with n elements A[1], A[2], ..., A[N] is in memory. The insertion sort algorithm scans A from A[1] to A[N], inserting each element A[K] into its proper position in the previously sorted subarray A[1], A[2], ..., A[K-1].

Example

Suppose an array A contains 8 elements as follows: 77, 33, 44, 11, 88, 22, 66, 55

PASS	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A
K=1		(77)	33	44	11	88	22	66	55
K=2		77	(33)	44	11	88	22	66	55
K=3		33	77	(44)	11	88	22	66	55
K=4		33	44	77	(11)	88	22	66	55
K=5		11	33	44	77	(88)	22	66	55
K=6		11	33	44	77	88	(22)	66	55
K=7		11	22	33	44	77	88	66	55
K=8		11	22	33	44	66	77	88	55
[illegible]		11	22	33	44	55	66	77	88

Date .....
Page .....

Algorithm: Insertion (A, N)

1. Set A[0] = $-\infty$ [Initializes with very small no]
2. Repeat Steps 3 to 5 for K = 2, 3, - - - N.
3. Set TEMP := A[K] & PTR := K-1
4. Repeat while TEMP < A[PTR]
   (a) Set A[PTR+1] := A[PTR]
   (b) Set PTR := PTR-1
   [End of loop]
5. Set A[PTR+1] := TEMP [Inserts element in proper place]
   [End of Step 2 loop]
6. Return

Complexity — Worst Case: $\frac{n(n-1)}{2} = O(n^2)$ — Average Case: $\frac{n(n-1)}{4} = O(n^2)$

(2) SELECTION SORT

Suppose an Array A with n elements A[1], A[2], ... A[N] is in memory. It works as follows:

First find the smallest element in the list & put it in the first position. Then find the second smallest element in list and put it in the second position.

Example 77, 33, 44, 11, 88, 22, 66, 55

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1, LOC=4	(77)	33	44	(11)	88	22	66	55
K=2, LOC=6	11	(33)	44	77	88	(22)	66	55
K=3 LOC=6	11	22	(44)	77	88	(33)	66	55
K=4 LOC=6	11	22	33	(77)	88	(44)	66	55
K=5 LOC=8	11	22	33	44	(88)	77	66	(55)
K=6 LOC=7	11	22	33	44	55	(77)	(66)	88
K=7 LOC=7	11	22	33	44	55	66	(77)	88
Sorted	11	22	33	44	55	66	77	88

Date
Page

Procedure: MIN(A, K, N, LOC)
This procedure finds the location LOC of the smallest element among A[K], A[K+1], ... A[N]

1. Set MIN := A[K] & LOC := K
2. Repeat for J = K+1, K+2, ... N
   If MIN > A[J] then
   Set MIN := A[J] & LOC := J
   [End of loop]
3. Return.

Algorithm: SELECTION(A, N)

1. Repeat steps 2 and 3 for K = 1, 2, ... N-1
2. Call MIN(A, K, N, LOC)
3. [Interchange A[K] & A[LOC]]
   Set TEMP = A[K], A[K] := A[LOC] & A[LOC] := TEMP
   [End of Step 1 loop]
4. Exit.

Complexity

Worst case: $\frac{n(n-1)}{2} = O(n^2)$

Average case: $\frac{n(n-1)}{2} = O(n^2)$

QUICK SORT

It follows divide & conquer strategy. Acc. to this algo. it is faster & easier to sort two small arrays than one larger one. It is also known as partition exchange sort.

1. Pick one element in the array as a pivot point
2. Make one pass through the array called a partition step, re-arranging the entries so that
   - the pivot is in its proper place.
   - entries smaller than the pivot are to the left of the pivot.
   - entries larger than the pivot are to its right.
3. Recursively apply quicksort to the part of the array that is to the left of the pivot & to the right part of the array.

Algorithm

1. Choose the pivot point
   → Take the median of first, last & middle element

Example 8, 3, 25, 6, 10, 17, 1, 2, 18, 5
Median of [8, 5, 10] is 8

2. Partitioning
   (a) The first action is to get the pivot out of the way – swap it with the last element

5, 3, 25, 6, 10, 17, 1, 2, 18, 8

(6) We want larger elements to go to the right and smaller elements to go to the left.

⇒* While i is to the left of j, we move i right, skipping all the elements less than the pivot. If an element is found greater than the pivot, i stops.

⇒* While j is to the right of i, we move j left skipping all the elements greater than pivot. If an element is found less than the pivot, j stops

⇒* When both i & j have stopped, the elements are swapped.

⇒* When i & j have ~~stopped~~ crossed, no swap is performed, scanning stops & the element pointed to by i is swapped with the pivot.

Example

i = 3 j = 8 swapped 25 — 2

5, 3, 25, 6, 10, 17, 1, 25, 18, 8

i = 5, j = 7 swapped 10 & 1

5, 3, 2, 6, 1, 17, 10, 25, 18, 8

(c) Restore pivot

i = 6, j = 6 crossed. swap 17, 8

[5, 3, 2, 6, 1] [8] [10, 25, 18, 17]

STEP3 Recursively quicksort the left & right p

Example

RIGHT : 5, 3, 2, 6, 1

↑ median

5, 3, 1, 6, 2

↑ i ↑ j

i=1, j=4 swap 5, 1

1, 3, 5, 6, 2

↑ i ↑ j

i=1, j=3 swap 5, 1

1, 3, 5, 6, 2

↑ i ↑ j

i=2, j=2 swap 3, 2

1, [2], [5, 6, 3]

↑

[1, 2, 3, 5, 6]

LEFT [10, 25, 18, 17]

Median 17 ↑ i ↑ j

i=2, j=i crossed swap 25, 17

[10, 17, 18, 25]

Result.

[1, 2, 3, 5, 6, 8, 10, 17, 18, 25]

2018/10/30 09:49

Date .....
Page .....

MERGE SORT

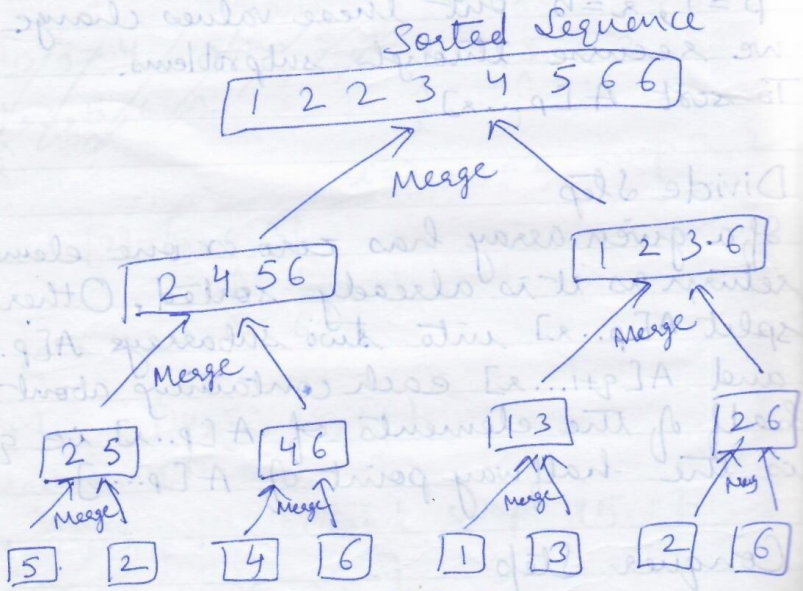
It is based on divide & conquer paradi
Sort a subarray A[p..r]. Initially
p=1, r=n but these values change a
we recurse through subproblems.
To sort A[p...r]

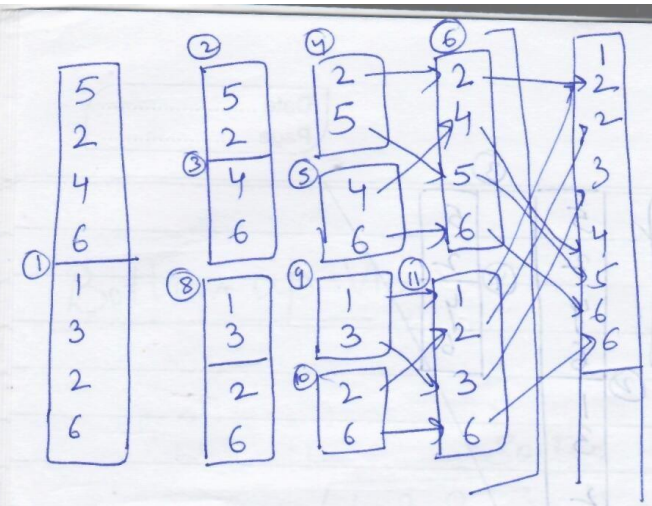
1. Divide Step
If a given array has zero or one elemen
return as it is already sorted. Otherwi
split A[p..r] into two subarrays A[p..q
and A[q+1..r] each containing about
half of the elements of A[p..r] i.e q
is the halfway point of A[p...r].

2. Conquer Step
Conquer by recursively sorting the
two subarrays A[p..q] & A[q+1..r]

3. Combine Step
Combine the elements back in A[1..r
by merging the two sorted subarrays
A[p...q] & A[q+1...r] into a sorted sequence
To accomplish this step, we will define
procedure MERGE[A, p, q, r]

Bottom up View





ee.ryerson.ca

① Dividing list in half 4+4
② Mergesort first half
  ↳ ③ Dividing list in half 2+2
    ↳ ④ Sorting first half
    ↳ ⑤ Sorting second half
    ↳ ⑥ Merging list
⑦ Mergesort second half
  ↳ ⑧ Dividing list in half 2+2
    ↳ ⑨ Sorting first half
    ↳ ⑩ Sorting second half
    ↳ ⑪ Merging list
⑫ Merging list.

Defenition
Recursively dividing the unsorted list in half merge sorting the left side & then right side & merging the right & left back together

Complexity $O(n \log n)$

Date
Page

Algorithm : MergeSort (A, p, r)

```
1. If p < r
2. then q = INTEGER ((p+q)/2)
3.      MergeSort (A, p, q)
4.      MergeSort (A, q+1, r)
5.      Merge (A, p, q, r)
```

NOTE : Intial Call MergeSort (A, 1, n)

Procedure : Merge (A, p, q, r)

```
(1)  n1 = q - p + 1
(2)  n2 = r - q
(3)  for i = 1 to n1
         do L[i] = A[p + i - 1]
(4)  for j = 1 to n2
         do R[j] = A[q + j]
(5)  L[n1 + 1] = -∞
(6)  R[n2 + 1] = -∞        } Setinel to avoid having to check if either subarray is fully copied at each step
(7)  i = 1, j = 1
(8)  for k = p to r
         do if L[i] ≤ R[j]
               then A[k] = L[i]
                    i = i + 1
               else A[k] = R[j]
                    j = j + 1
```

RADIX SORT

This algo. puts the elements in order by comparing the digits of the nos.

Example

Consider the following nos

493, 812, 715, 710, 195, 437, 582, 340, 385

① Start comparing & ordering the one's digit

Digit	Sublist
0	710, 340
1	
2	812, 582
3	493
4	
5	715, 195, 385
6	
7	437
8	
9	

→ No. were added onto the list in the order that they were found, which is why nos appear to be unsorted in each of the sublists above.

→ Now gather the sublist into the main list again

710, 340, 812, 582, 493, 715, 195, 385, 437.

② Create sublist again based on ten's digit

2018/10/30 09:50

Digit	Sublist
0	
1	710, 812, 715
2	
3	437
4	340
5	
6	
7	
8	582, 385
9	493, 195

→ Gather sublist again

710, 812, 715, 437, 340, 582, 385, 493, 195

③ Finally sublists are created according to the 100's digit

Digit	Sublist
0	
1	195
2	
3	340, 385
4	437, 493
5	582
6	
7	710, 715
8	812
9	

~~At last, the list is gathered upon~~

195, 340, 385, 437, 493, 582, 710, 715, 812

Algo.

1. In the first iteration the elements are picked up & kept in various pockets che their unit's digit
2. The cards are collected from pocket & again they are given as input to
3. In second iteration, the ten's digit sorted.
4. Repeat through step 2.
5. In the final iteration, the digit at he position are sorted
6. Repeat through step 2.

Complexity $O(n*n)$

n → loop traverse each digit.

n → inner loop for each elem