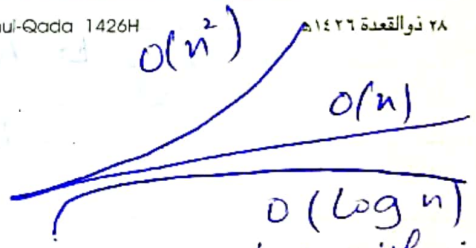
Data Structures & Algorithms :-

① Big O Notation. → Linear growth
① O(n) ⇒ Constants are dropped and the cost of algorithm increases linearly.
↳ Predicts the execution time for an algorithm or vice versa
↳ Constant-time, runtime complexity.
↳ It also predicts the complexity of software its scalability and performance when it comes to large datsets.

② $O(n^2)$ → Quadratic growth
↳ Can't handle large dataset cause the speed decreasc of algorithms. Cause they are increasing quadratically.
↳ So as the power of n which is 2 increases it gets more and more slow.



ملاحظات NOTES ملاحظات NOTES ملاحظات

31 SATURDAY السبت | ٣١ | 1 SUNDAY الأحد | ١ | 2 MONDAY الاثنين | ٢

29 Thul-Qada 1426H | ٢٩ ذو القعدة ١٤٢٦هـ | 1 Thul-hejaa 1426H | ١ ذو الحجة ١٤٢٦هـ | 2 Thul-hejaa 1426H | ٢ ذو الحجة ١٤٢٦هـ

③ $O(\log n)$ => Logarithmic growth

↳ Logarithmic growth slows with increase of input size.

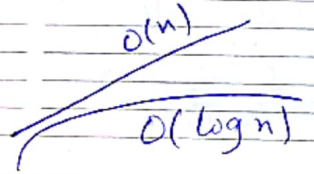
↳ Slows down at some time point

↳ Algorithms running on this factor are more efficient and more scalable than the linear one which is $O(n)$

↳ Uses binary search

↳ Reduces work by half.

↳ Algorithm that uses logarithmic time is more scalable than the linear one.



NOTES ملاحظات | NOTES ملاحظات | NOTES ملاحظات

3 TUESDAY الثلاثاء | ٣ | 4 WEDNESDAY الأربعاء | ٤ | 5 THURSDAY الخميس | ٥

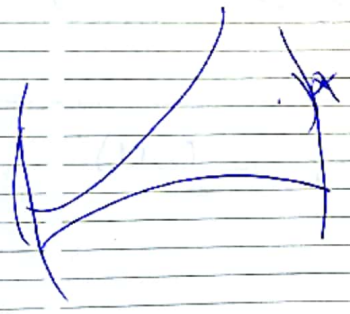
3 Thul-hejaa 1426H | ٣ ذو الحجة ١٤٢٦هـ | 4 Thul-hejaa 1426H | ٤ ذو الحجة ١٤٢٦هـ | 5 Thul-hejaa 1426H | ٥ ذو الحجة ١٤٢٦هـ

④ $O(2^n)$ => Exponential growth

↳ Quite opposite of logarithmic growth.

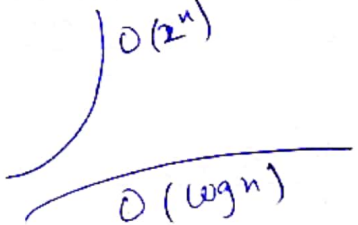
↳ As input size increases the curve rises faster.

↳ Algorithm running in exponential curve/time is not scalable and becomes very slow very soon.



6 FRIDAY الجمعة | ٦

6 Thul-hejaa 1426H | ٦ ذو الحجة ١٤٢٦هـ



NOTES ملاحظات | NOTES ملاحظات | NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F

MARCH مارس/آذار

S	S	M	T	W	T	F

APRIL أبريل/نيسان

S	S	M	T	W	T	F

MAY مايو/أيار

S	S	M	T	W	T	F

JUNE يونيو/حزيران

S	S	M	T	W	T	F

JULY يوليو/تموز

S	S	M	T	W	T	F

AUGUST أغسطس/آب

S	S	M	T	W	T	F

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F

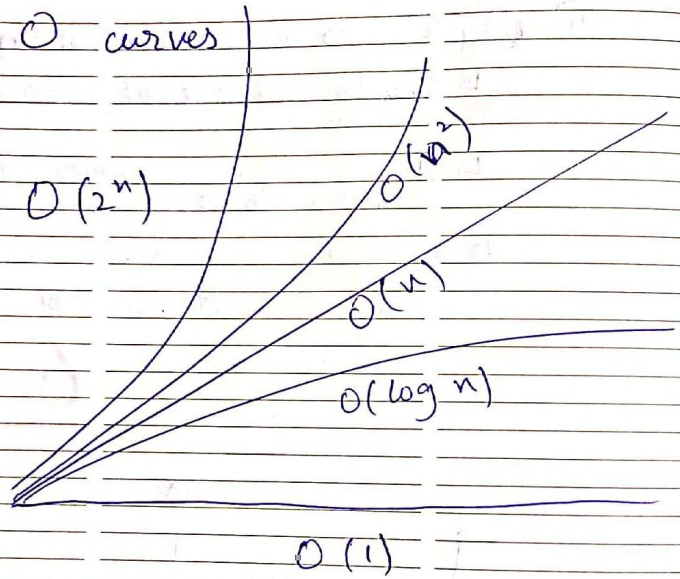
DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F

7 SATURDAY السبت | 8 SUNDAY الأحد | 9 MONDAY الاثنين

7 Thul-hejaa 1426H ٧ ذوالحجة ١٤٢٦ | 8 Thul-hejaa 1426H ٨ ذوالحجة ١٤٢٦ | 9 Thul-hejaa 1426H ٩ ذوالحجة ١٤٢٦

⑤ Big O curves



NOTES ملاحظات

10 TUESDAY الثلاثاء | 11 WEDNESDAY الأربعاء | 12 THURSDAY الخميس

10 Thul-hejaa 1426H ١٠ ذوالحجة ١٤٢٦ | 11 Thul-hejaa 1426H ١١ ذوالحجة ١٤٢٦ | 12 Thul-hejaa 1426H ١٢ ذوالحجة ١٤٢٦

⑥ Space-time complexity, runtime complexity :-

O(n)

↳ Spacetime for limited space in order to preserve the memory.

13 FRIDAY الجمعة

13 Thul-hejaa 1426H ١٣ ذوالحجة ١٤٢٦

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

JULY يوليو/تموز

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

14 SATURDAY السبت ١٤ | 15 SUNDAY الأحد ١٥ | 16 MONDAY الاثنين ١٦

14 Thul-hejaa 1426H ١٤ ذوالحجة ١٤٢٦ | 15 Thul-hejaa 1426H ١٥ ذوالحجة ١٤٢٦ | 16 Thul-hejaa 1426H ١٦ ذوالحجة ١٤٢٦

② Arrays :-
↳ storing
↳ List of items, numbers, objects etc in a sequential way.
↳ Runtime complexity of O(1) for simple addition or deletion of an element from the inden of ~~an~~ array.
↳ Runtime complexity of O(n) for complex addition or deletion of an element from start or from the end of array, which costs for filling up the blank space. for that we use linked list
↳ lookup uses O(1) runtime complexity which is constant

=> Runtime complexities
lookup by inden O(1)
lookup by value O(n)
Insertion O(n)
Deletion O(n)

NOTES ملاحظات
↳ Arrays don't grow or shrink automatically like in static language like java C#, we use array list for that scenario to initialize that array list class.

17 TUESDAY الثلاثاء ١٧ | 18 WEDNESDAY الأربعاء ١٨ | 19 THURSDAY الخميس ١٩

17 Thul-hejaa 1426H ١٧ ذوالحجة ١٤٢٦ | 18 Thul-hejaa 1426H ١٨ ذوالحجة ١٤٢٦ | 19 Thul-hejaa 1426H ١٩ ذوالحجة ١٤٢٦

③ Linked lists :-
↳ Grow and shrinks automatically.
↳ Store list of objects in sequence.
↳ Group of nodes in a sequence.
↳ Nodes consists of ~~two~~ two parts of data, one is value and other is address for the reference to next node.
↳ First node => Head
↳ Last node => Tail
↳ Runtime complexity ~~of~~ from Head to tail. => O(n) or by value but by inden its O(n)

20 FRIDAY الجمعة ٢٠

20 Thul-hejaa 1426H ٢٠ ذوالحجة ١٤٢٦

NOTES ملاحظات
↳ In insertion at end its => O(1) and also same as with the start or head => O(1)
↳ But if insertion operation and the updation is in middle its O(n)

JANUARY يناير/كانون الثاني 2006/ ٢٠٠٦

21 SATURDAY السبت ٢١ | 22 SUNDAY الأحد ٢٢ | 23 MONDAY الاثنين ٢٣

21 Thul-hejja 1426H | ٢١ ذوالحجة ١٤٢٦ | 22 Thul-hejja 1426H | ٢٢ ذوالحجة ١٤٢٦ | 23 Thul-hejja 1426H | ٢٣ ذوالحجة ١٤٢٦

↳ For deletion ~~at~~ in linked list from head with O(1) operation.
↳ But for deletion from tail we will use O(n) operation.
↳ But for middle ~~insertion~~ deletion O(n) operation will be used again.

⇒ Arrays vs linked lists:-

↳ Arrays have fixed size. (such as static arrays)
↳ for more memory or input values Dynamic arrays are used but they occupy upto 50-100% of the memory and waste alot of it
↳ Linked lists only take memory as it is required to store the values

NOTES ملاحظات

↳ They take the memory to store the address of the next and previous node
↳ Arrays have smaller footprint and to store values which are not confirmed how many, dynamic ones are used.

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
					1	2
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

JANUARY يناير/كانون الثاني 2006/ ٢٠٠٦

24 TUESDAY الثلاثاء ٢٤ | 25 WEDNESDAY الأربعاء ٢٥ | 26 THURSDAY الخميس ٢٦ | 27 FRIDAY الجمعة ٢٧

24 Thul-hejja 1426H | ٢٤ ذوالحجة ١٤٢٦ | 25 Thul-hejja 1426H | ٢٥ ذوالحجة ١٤٢٦ | 26 Thul-hejja 1426H | ٢٦ ذوالحجة ١٤٢٦ | 27 Thul-hejja 1426H | ٢٧ ذوالحجة ١٤٢٦

↳ But it all depends on the certain problem that one tackles with.
↳ Some operations are higher in arrays but slower in linked lists.

⇒ Runtime complexity of Arrays vs linked list:-

	Arrays	Linked Lists
Lookup:		
By index	O(1)	O(n)
By value	O(n)	O(n)
Insert:		
start/end	O(n)	O(1)
middle	O(n)	O(n)
Delete:		
Beginning	O(n)	O(1)
Middle	O(n)	O(n)
End	O(n)	O(n)

NOTES ملاحظات

JULY يوليو/تموز

S	S	M	T	W	T	F
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

JANUARY يناير/كانون الثاني 2006/ ٢٠٠٦

28 SATURDAY السبت ٢٨ 29 SUNDAY الأحد ٢٩ 30 MONDAY الاثنين ٣٠

=> Types of linked lists

↳ Singly..

↳ Traditional linked list

↳ Every node has the reference or the pointer to the next node.

↳ like reference

↳ sequential.

↳ Doubly:-

↳ Each node has the reference or the pointer to its previous node too.

↳ forward / backward

FEBRUARY فبراير/شباط 2006/ ٢٠٠٦

31 TUESDAY الثلاثاء ٣١ 1 WEDNESDAY الأربعاء ١ 2 THURSDAY الخميس ٢

3 FRIDAY الجمعة ٣

↳ Singly:

↳ Deleting from end costs O(n) operation

↳ It takes less memory

↳ As each node has a single field to hold the address of the next node

↳ But didn't give the performance gain

↳ Doubly:

↳ Deleting from end costs O(1) operation

↳ Takes more memory than the singly linked lists

↳ As it holds two fields of next and previous nodes

↳ But gives a performance gain.

NOTE:- Both singly and doubly can be circular just like music player list

4 SATURDAY السبت | ٤ 5 SUNDAY الأحد | ٥ 6 MONDAY الاثنين | ٦

5 Muharram 1427H | ٥ محرم ١٤٢٧ | 6 Muharram 1427H | ٦ محرم ١٤٢٧ | 7 Muharram 1427H | ٧ محرم ١٤٢٧

Note:-
pointers are variables / that stores objects
the values of the ~~next~~ other variables addresses.

Summary:-
↳ linked lists automatically grows and shrinks without the loss of memory. But still requires a little more memory.

⇒ Runtime complexity:-
lookup:-
By index O(n)
By value O(n)

Insert:-
Beginning / End O(1)
Middle O(n)

Delete:-
Beginning O(1)
Middle O(n)
End O(1) / O(n)
doubly
singly

7 TUESDAY الثلاثاء | ٧ 8 WEDNESDAY الأربعاء | ٨ 9 THURSDAY الخميس | ٩

8 Muharram 1427H | ٨ محرم ١٤٢٧ | 9 Muharram 1427H | ٩ محرم ١٤٢٧ | 10 Muharram 1427H | ١٠ محرم ١٤٢٧

④ Stacks:-
↳ Implement undo features in app
↳ Build compilers (e.g. syntax checking)
↳ Evaluate arithmatic expressions
↳ Build navigation systems / apps i.e. forward / backward.
↳ Like stack of books which uses LIFO (Last in and first out) principle
↳ It is a wrapper around arrays and linkedlists.
↳ Operations:-
push(item) (adds item on the top)
pop (item) (removes item on the top)
peek() (return item on the top without removing it from stack)
isEmpty() (to tell stack is empty or not)

O(1) ← quickly insert item in Linkedlist
O(1) ← constant time

10 FRIDAY الجمعة | ١٠

11 Muharram 1427H | ١١ محرم ١٤٢٧

11 SATURDAY السبت ١١ | 12 SUNDAY الأحد ١٢ | 13 MONDAY الاثنين ١٣

12 Muharram 1427H | 13 Muharram 1427H | 14 Muharram 1427H

↳ Stacks are not meant for storing list of objects and look them up quickly. So 'search' method never used.

↳ Note:
String is immutable, can't be modified so they can be expensive as with every string calling function a new string object is created in the memory.

⇒ Summary:-

↳ Stacks acts according LIFO principle.

NOTES ملاحظات

↳ They are implemented using arrays / linked lists.

↳ They run in O(1) operation.

↳ All the operations of stacks follows O(1) complexity

14 TUESDAY الثلاثاء ١٤ | 15 WEDNESDAY الأربعاء ١٥ | 16 THURSDAY الخميس ١٦

15 Muharram 1427H | 16 Muharram 1427H | 17 Muharram 1427H

⑤ Queues:-

↳ Apps based on order/view sequence.

↳ Similar to stack, but first item inserted is to be removed first so it is FIFO. (can be)

↳ Works in a sequential way.

17 FRIDAY الجمعة ١٧ | 18 Muharram 1427H

↳ For example a resource used by different/shared that resource one by one but in a queue.

↳ Printers, OS, Webservers, live support system.

↳ Operations:-

Runtime Complexities

O(1) ↳ enque (adding item at the back of queue)

O(1) ↳ deque (removing item from the front end of the queue)

O(1) ↳ peek (getting the item at front without removing it)

O(1) ↳ IsEmpty, IsFull (either queue is full or empty).

NOTES ملاحظات

18 SATURDAY السبت ١٨ | 19 SUNDAY الأحد ١٩ | 20 MONDAY الاثنين ٢٠

19 Muharram 1427H ١٩ محرم ١٤٢٧ | 20 Muharram 1427H ٢٠ محرم ١٤٢٧ | 21 Muharram 1427H ٢١ محرم ١٤٢٧

Note: Interfaces don't code like classes so they can be instantiated

Note:- "Array Deque" in this type of array one can add item into an array from any direction, using resizeable array class.

Note: Just adding the items into the queue costs O(1) operation while "dequeue" or removing costs O(n) operation respectively.

↳ Priority Queues:-
- ↳ Objects are processed on the basis of priorities rather than joining sequences.
- ↳ Numbers/objects are added automatically in a order-format instead of being added randomly.
- ↳ It uses O(n) operations as items are sorted through out the process.

NOTES ملاحظات

21 TUESDAY الثلاثاء ٢١ | 22 WEDNESDAY الأربعاء ٢٢ | 23 THURSDAY الخميس ٢٣

22 Muharram 1427H ٢٢ محرم ١٤٢٧ | 23 Muharram 1427H ٢٣ محرم ١٤٢٧ | 24 Muharram 1427H ٢٤ محرم ١٤٢٧

(6) Hash Tables:-
- ↳ ~~Know~~ AKa Dictionaries.
- ↳ Superfast lookups
- ↳ Spell checkers
- ↳ Building language Dictionaries.
- ↳ Compilers to lookup address of functions and variables.
- ↳ Code Editors.
- ↳ Called in different languages

24 FRIDAY الجمعة ٢٤

25 Muharram 1427H ٢٥ محرم ١٤٢٧

JAVA	JS	PYTHON	C#
↓	↓	↓	↓
HashMap	Object	Dictionary	Dictionary

↳ These are used to store key-value pairs. e.g key (employee no.), value (employee).
- ↳ The process is simple, To store an employee number in a hashtable, hashtable takes the employee object and store its address in the memory using hashfunction. hashfunction tells where to store that value in the memory.

NOTES ملاحظات

25 SATURDAY السبت ٢٥ | 26 SUNDAY الأحد ٢٦ | 27 MONDAY الاثنين ٢٧

26 Muharram 1427H | 27 Muharram 1427H | 28 Muharram 1427H

↳ e.g

employee no. → hash function → address in the memory

↳ This hash function is deterministic.
i.e giving the same input into it, gives ~~out~~ the same ouput onto us.

↳ Hash function is used for both storing and retrival of data or objects

↳ Operations

Runtime Complexity

O(1) ↳ Insert

O(1) ↳ Lookup

O(1) ↳ Delete/remove.

↳ Sets:

NOTES ملاحظات

↳ They have Key and not values

↳ They don't allow duplicate Keys like maps.

↳ generates unique Keys.

28 TUESDAY الثلاثاء ٢٨ | 1 WEDNESDAY الأربعاء ١ | 2 THURSDAY الخميس ٢

29 Muharram 1427H | 1 Safar 1427H | 2 Safar 1427H

↳ Index value using hash function is the value of the memory where the objects/data is stored in the memory.

↳ Hash value, the numeric presentation of the object/data on the memory. hashcode() or etc.

~~Related~~ representable by

3 FRIDAY الجمعة ٣

3 Safar 1427H

↳ Working of hash function

↳ Collision

Key1 → ○ Hash function → Address

Key2 →

↳ Two distinct Keys generated same hash value

↳ Two values that are colliding

NOTES ملاحظات

↳ Ways to handle collision.

① ↳ Pointing each cell in an array to a linked list.

↳ While having collision just add new item at the end of linked list. also 'chaining'

MARCH مارس/آذار 2006/ ٢٠٠٦

4 SATURDAY السبت ٤ | 5 SUNDAY الأحد ٥ | 6 MONDAY الاثنين ٦

4 Safar 1427H | 5 Safar 1427H | 6 Safar 1427H

② Finding different slots for storing those distinct values aka "open addressing" or finding different address to store second value.

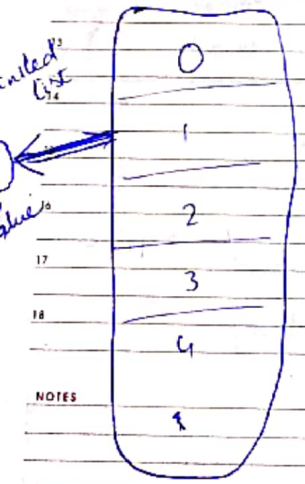
① Chaining:-

↳ Storing values by wrapping them in a linked list node

↳ This is done by HashTable which has the limit/array of five cells aka buckets/slots.

↳ Using Key value pair. Key is passed through hash function which works by the modulus of Key % to the array of hashtable as;

Key % Array(hashtable) = index of hash table.



NOTES ملاحظات

MARCH مارس/آذار 2006/ ٢٠٠٦

7 TUESDAY الثلاثاء ٧ | 8 WEDNESDAY الأربعاء ٨ | 9 THURSDAY الخميس ٩

7 Safar 1427H | 8 Safar 1427H | 9 Safar 1427H

② Open Adressing

↳ Linear probing:-

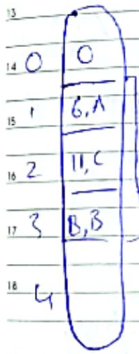
↳ We have to go through probing (searching) for addressing a same Key-value pair into a different slot.

↳ hash(Key) = Key % table-size.

↳ In linear we find for empty slot linearly

linear probing : hash(Key) + i

loop variable starts with 0/null until to find empty slot.



↳ values stored next to each other forming a 'cluster' which causes problems with probing causing it slower as the cluster grows.

↳ these clusters causes a drastic decrease in performance of the algorithm.

10 FRIDAY الجمعة ١٠

10 Safar 1427H

NOTES ملاحظات

MARCH مارس/آذار 2006/ ٢٠٠٦

11 SATURDAY السبت ١١ | 12 SUNDAY الأحد ١٢ | 13 MONDAY الاثنين ١٣

11 Safar 1427H | 12 Safar 1427H | 13 Safar 1427H

① Quadratic Probing

↳ hash(Key) + i^2 ~~(hash(Key)~~

As i is doubled so the Key-value pair will spread out and stop the formation of clusters hash table.

↳ (hash(Key) + i)^2 % table_size / array size

↳ in order to reduce the 'i' to the boundry of the array if it grows beyond the boundry of array.

↳ But Quadratic Probing one could find it difficult to find the ~~Key-value pair~~ the empty slot ~~top~~ of array cause it might fall through infinite iterations / loop

NOTES ملاحظات

14 TUESDAY الثلاثاء ١٤ | 15 WEDNESDAY الأربعاء ١٥ | 16 THURSDAY الخميس ١٦ | 17 FRIDAY الجمعة ١٧

14 Safar 1427H | 15 Safar 1427H | 16 Safar 1427H

③ Double Hashing :-

↳ Uses independant hash function to calculate no. of steps.

↳ Double. i x hash2

↳ hash2(Key) = prime - (Key % prime)

↳ 2nd hash function

↳ For index :-

(hash1(Key) + i x (hash2(Key)) % table_size

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

JULY يوليو/تموز

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

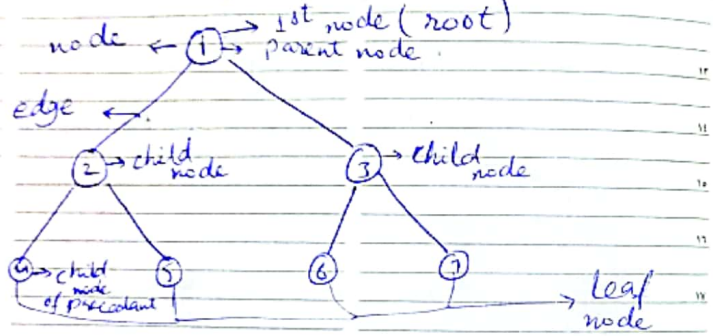
S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

18 SATURDAY السبت ١٨ | 19 SUNDAY الأحد ١٩ | 20 MONDAY الاثنين ٢٠

18 Safar 1427H | 19 Safar 1427H | 20 Safar 1427H

⑦ Trees :-

↳ Databases, websites, GUI's
↳ Stores elements in hierarchy.
↳ Elements are nodes while lines connecting them are edges.
↳ Nodes (stores) ⇒ objects, values, data.
↳



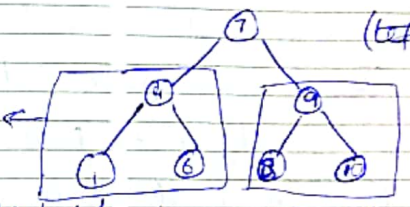
↳ Above is binary tree as one is having two child node.
↳ A node without any children is leaf node
↳ Representing hierarchial data
↳ DBMS for indexing of data.
↳ Autocomplete like writing search query in a search engine/browser.

NOTES ملاحظات

21 TUESDAY الثلاثاء ٢١ | 22 WEDNESDAY الأربعاء ٢٢ | 23 THURSDAY الخميس ٢٣

21 Safar 1427H | 22 Safar 1427H | 23 Safar 1427H

↳ Compilers, using syntax tree to parse expression.
↳ Compression algorithms with format of .jpeg and .mp3.
↳ Binary search tree



(~~left child node < parent node~~) ×

24 FRIDAY الجمعة ٢٤

24 Safar 1427H

leftchildnode < parent node < rightchild node.
↳ As goes same with the subtrees.

↳ Logarithmic time complexity.
Throwing out half of items and narrowing down search like in above binary tree searching from smaller node value to the target node value which is '7' while ignoring higher node values like '9,8,10,' and narrowing to compare with only '4' and '7'.

NOTES ملاحظات

JANUARY يناير/كانون الثاني
FEBRUARY فبراير/شباط
MARCH مارس/آذار
APRIL أبريل/نيسان
MAY مايو/أيار
JUNE يونيو/حزيران
JULY يوليو/تموز
AUGUST أغسطس/آب
SEPTEMBER سبتمبر/أيلول
OCTOBER أكتوبر/تشرين الأول
NOVEMBER نوفمبر/تشرين الثاني
DECEMBER ديسمبر/كانون الأول

25 SATURDAY السبت ٢٥ | 26 SUNDAY الأحد ٢٦ | 27 MONDAY الاثنين ٢٧

↳ Operations:

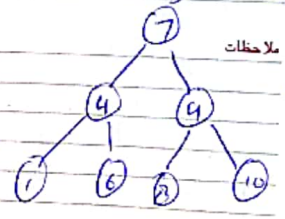
	Runtime Complexity
↳ LookUp	O(log n)
↳ Insert	O(log n)
↳ Delete	O(log n)

↳ If trees not structured properly then performance might degrade to the runtime complexity of O(n)

↳ Trees are much better than arrays, linked lists in performance wise.

↳ Non-linear structures

↳ Breadth first: (level order) traversal. e.g: instead of going linearly through elements like arrays and linked lists it goes in through levels like



ملاحظات NOTES → 7, 4, 9, 1, 6, 8, 10

28 TUESDAY الثلاثاء ٢٨ | 29 WEDNESDAY الأربعاء ٢٩ | 30 THURSDAY الخميس ٣٠ | 31 FRIDAY الجمعة ٣١

↳ Depth First traversal

↳ Pre-order Root → left, right

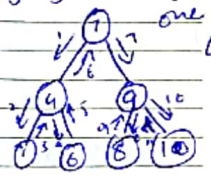
↳ In-order left, root, right

↳ Post-order left, right, root.

↳ Instead of going level to level search/traversal one can go deep from root/parent to other node (children) and leaf node (grandchildren)

↳ Pre-order →

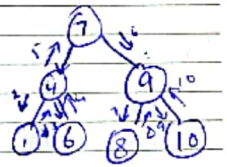
7, 4, 1, 6, 9, 8, 10



↳ In-order →

1, 4, 6, 7, 8, 9, 10

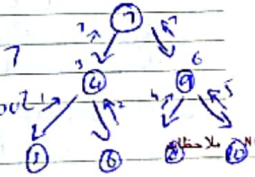
left, root, right
or
right, root, left



↳ Post-order →

1, 6, 4, 8, 10, 9, 7

↳ From leaf to root



↳ Depth first traversal can be done easily through recursion.

Note:
Recursion means when a function or a method calls ~~itself~~ itself

Note:
When calling a recursion one should keep in mind have a base condition in loop or code in order to avoid recursion from executing or running forever or crashing.

1 SATURDAY السبت — 12 SUNDAY الأحد — 23 MONDAY الاثنين — ٣

3 Rabi' I 1427H — ٣ ربيع الأول ١٤٢٧هـ — 4 Rabi' I 1427H — ٤ ربيع الأول ١٤٢٧هـ — 5 Rabi' I 1427H — ٥ ربيع الأول ١٤٢٧هـ

↳ Calculating the height of tree/node
↳ 1 + max (height(L), height(R))
↳ Starts from the deep/depth leaves and end at root/parent
↳ while ~~de~~ to calculate depth of a tree one starts from root uptill the leaf of the tree.
↳ Minimum value in binary tree:-
↳ Binary search tree:-
↳ All values in the left sub tree are small
↳ All values in the righ sub tree of the node is greater than the node itself.
↳ Through this algorithm we are visity each node only once.
↳ Nodes at Kth distane
↳ Distance from the root node to the leaf node or the target node.
↳ This distance decrements from top to bottom eg (3→0)

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	M	T	W	T	F	S
1	2	3	4	5	6	
8	9	10	11	12	13	
15	16	17	18	19	20	
22	23	24	25	26	27	
29	30	31				

FEBRUARY فبراير/شباط

S	M	T	W	T	F	S
			1	2	3	
5	6	7	8	9	10	
12	13	14	15	16	17	
19	20	21	22	23	24	
26	27	28				

MARCH مارس/آذار

S	M	T	W	T	F	S
			1	2	3	
5	6	7	8	9	10	
12	13	14	15	16	17	
19	20	21	22	23	24	
26	27	28	29	30	31	

APRIL أبريل/نيسان

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	
9	10	11	12	13	14	
16	17	18	19	20	21	
23	24	25	26	27	28	
30						

MAY مايو/أيار

S	M	T	W	T	F	S
		1	2	3	4	5
7	8	9	10	11	12	
14	15	16	17	18	19	
21	22	23	24	25	26	
28	29	30	31			

JUNE يونيو/حزيران

S	M	T	W	T	F	S
				1	2	
4	5	6	7	8	9	
11	12	13	14	15	16	
18	19	20	21	22	23	
25	26	27	28	29	30	

4 TUESDAY الثلاثاء — 45 WEDNESDAY الأربعاء — 56 THURSDAY الخميس — ٦

6 Rabi' I 1427H — ٦ ربيع الأول ١٤٢٧هـ — 7 Rabi' I 1427H — ٧ ربيع الأول ١٤٢٧هـ — 8 Rabi' I 1427H — ٨ ربيع الأول ١٤٢٧هـ

↳ level order traversal (Breadth first)
↳ Levels = height of tree
↳ From leaf to root height increases
↳ height of tree = height of root node
↳ height of root node = number of edges ~~to~~ in the longest path to a leaf node.
↳ That height of the tree can be used to print each level using for loop.
↳ Depth increases from top to bottom
↳ height increases from bottom to top.

7 FRIDAY الجمعة — ٧

9 Rabi' I 1427H — ٩ ربيع الأول ١٤٢٧هـ

NOTES ملاحظات

JULY يوليو/تموز

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	
9	10	11	12	13	14	
16	17	18	19	20	21	
23	24	25	26	27	28	
30	31					

AUGUST أغسطس/آب

S	M	T	W	T	F	S
			1	2	3	4
6	7	8	9	10	11	
13	14	15	16	17	18	
20	21	22	23	24	25	
27	28	29	30	31		

SEPTEMBER سبتمبر/أيلول

S	M	T	W	T	F	S
						1
3	4	5	6	7	8	
10	11	12	13	14	15	
17	18	19	20	21	22	
24	25	26	27	28	29	

OCTOBER أكتوبر/تشرين الأول

S	M	T	W	T	F	S
	1	2	3	4	5	6
8	9	10	11	12	13	
15	16	17	18	19	20	
22	23	24	25	26	27	
29	30	31				

NOVEMBER نوفمبر/تشرين الثاني

S	M	T	W	T	F	S
			1	2	3	
5	6	7	8	9	10	
12	13	14	15	16	17	
19	20	21	22	23	24	
26	27	28	29	30		

DECEMBER ديسمبر/كانون الأول

S	M	T	W	T	F	S
31						1
3	4	5	6	7	8	
10	11	12	13	14	15	
17	18	19	20	21	22	
24	25	26	27	28	29	

⑧ AVL Trees :-

Note:-

↳ Balanced tree :-

~~(height (left subtree/node) - height)~~ *

height (left subtree/node) - height (right subtree/node) <= 1.

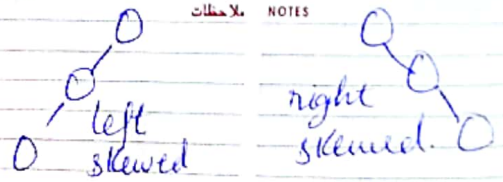
↳ Can't have long edges/branches.

↳ Unbalanced tree :-

↳ Trees become unbalanced due to the unregular addition of the nodes i.e. height of left subtree ≠ (not equal) to the height of right subtree.

↳ Right and left skewed binary trees are examples for unbalanced trees.

ملاحظات NOTES



↳ Trees become unbalanced when sorted values are added.

↳ Self-balancing Trees :-

↳ AVL Trees (Adelson-Velsky and Landis)

↳ Tree balances itself everytime a value is inserted or removed.

↳ Red-black Trees

↳ B-trees.

↳ Splay Trees

↳ 2-3 Trees.

14 FRIDAY الجمعة ١٤

16 Rabi' I 1427H | ١٦ ربيع الأول ١٤٢٧هـ

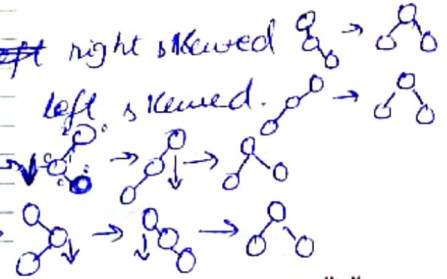
↳ AVL Rotations

↳ Left (LL) when right skewed

↳ Right (RR) when left skewed.

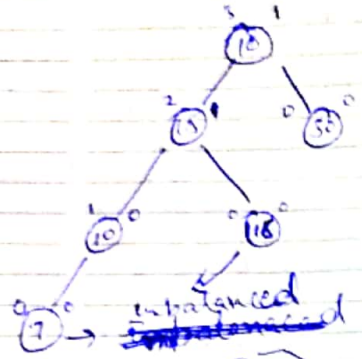
↳ Left-right (LR)

↳ Right-left (RL)

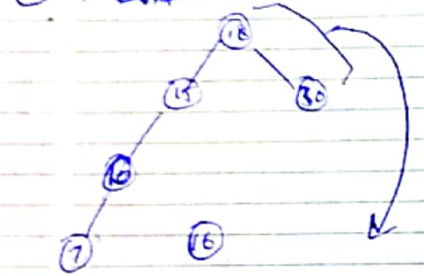


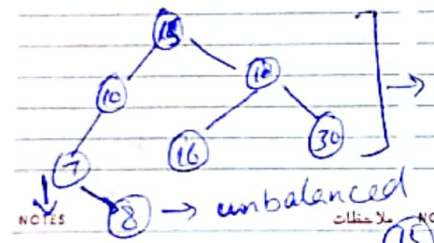
↳ By putting weight on the unbalanced target node.

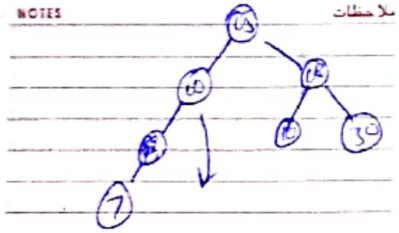
15 SATURDAY السبت ١٥ | 16 SUNDAY الأحد ١٦ | 17 MONDAY الاثنين ١٧

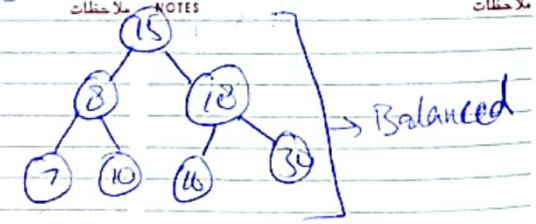


(RL)









18 TUESDAY الثلاثاء ١٨ | 19 WEDNESDAY الأربعاء ١٩ | 20 THU

~~Implementing a heap.~~

(9) HEAPS:-

↳ Used in sorting and algorithm.

↳ Its type of tree

↳ With properties like

↳ Complete tree:

↳ Every level except the last level is completely filled

↳ level-filling takes left to right.

↳ Completely filled with

↳ (2) Heap property:-

↳ Value of every node than or equal to its

↳ Max-heap: where root largest value, while while going down

↳ Min-heap: where root the smaller value.

↳ A binary Tree is a bi

↳ Heap is a complete binary tree

21

NOTES ملاحظات

22 SATURDAY السبت ٢٢ | 23 SUNDAY الأحد ٢٣ | 24 MONDAY الاثنين ٢٤

24 Rabi' I 1427H ٢٤ ربيع الأول ١٤٢٧هـ | 25 Rabi' I 1427H ٢٥ ربيع الأول ١٤٢٧هـ | 26 Rabi' I 1427H ٢٦ ربيع الأول ١٤٢٧هـ

↳ Applications:-

↳ Sorting data (heap sort)

↳ Graph algorithms (shortest path between two nodes) e.g. GPS.

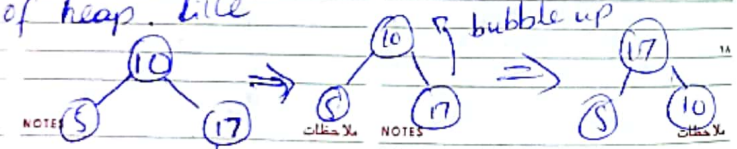
↳ Implementation of priority queues

↳ Finding out Kth smaller/largest value

↳ Operations:-

↳ children of the root node should be smaller than or equal to root node in max-heap while opposite in min-heap

↳ If children nodes are bigger or get bigger, we use 'bubbling up' operation of heap. like



↳ 'Bubbling down' works same as the 'bubbling up' but only when a root node is deleted and and it's a max-heap.

Note -
↳ The longest the root node travel down equals to the height of the tree which is equal to the log(n).

↳ Runtime Complexity

↳ It means the height of a balanced binary tree with 'n' nodes.

$h = \log(n)$

So,

$O(\log n)$ or $O(h)$

↳ Deletion of largest ↳ Remove node/insert

$O(\log n)$

↳ Finding max-value:

$O(1)$

↳ Finding min-value:

$O(1)$

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28		

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUNE يونيو/حزيران

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

25 TUESDAY الثلاثاء ٢٥ | 26 WEDNESDAY الأربعاء ٢٦ | 27 THURSDAY الخميس ٢٧

27 Rabi' I 1427H ٢٧ ربيع الأول ١٤٢٧هـ | 28 Rabi' I 1427H ٢٨ ربيع الأول ١٤٢٧هـ | 29 Rabi' I 1427H ٢٩ ربيع الأول ١٤٢٧هـ

↳ Implementing a heap:

↳ An array is used in its implementation even it's a binary tree, because heap trees don't have any holes.

↳ Due to this heaps have smaller footprint or space in the memory.

↳ Formula's for calculating indexes in heap

↳ For right children or sub tree or leaf node:

right = parent node × 2 + 2

↳ For left children or subtree or leaf node

left = parent node × 2 + 1

↳ For parent of a node:-

parent = (index - 1)/2

↳ For last Parent of sub tree:-

(last parent = (n/2) - 1

28 FRIDAY الجمعة ٢٨

30 Rabi' I 1427H ٣٠ ربيع الأول ١٤٢٧هـ

NOTES ملاحظات

↳ Heapify algorithm: transforming an array into a heap in place.

JULY يوليو/تموز

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

29 SATURDAY السبت ٢٩ | 30 SUNDAY الأحد ٣٠ | 1 MONDAY الاثنين ١

1 Rabi' II 1427H | ٢٩ ربيع الثاني ١٤٢٧ | 2 Rabi' II 1427H | ٣٠ ربيع الثاني ١٤٢٧ | 3 Rabi' II 1427H | ١ ربيع الثاني ١٤٢٧

(10) Tries:-

↳ Overlooped datastructures

↳ Tries derived from retrieval.

↳ Another type of trees but not binary tree, each child have several nodes.

↳ Other names are:

↳ Digital
↳ Radix
↳ Prefix trees.

↳ Used mostly in autocompletion like typing a word in google.

↳ Arrays of ~~words~~ strings is not used instead because a large number of queries or words are gonna take a large amount of space in the memory. as most of the words have same prefix.

↳ Tries doesn't duplicate data/words as the arrays does.

↳ like consider 'b' as a prefix for the words 'bag' and 'boy' so its better to store it in tries tree as it will have many child nodes in it. So one node can easily save upto many words as its child nodes.

NOTES ملاحظات

2 TUESDAY الثلاثاء ٢ | 3 WEDNESDAY الأربعاء ٣ | 4 THURSDAY الخميس ٤

4 Rabi' II 1427H | ٤ ربيع الثاني ١٤٢٧ | 5 Rabi' II 1427H | ٥ ربيع الثاني ١٤٢٧ | 6 Rabi' II 1427H | ٦ ربيع الثاني ١٤٢٧

↳ Operations.

↳ Lookups: (O(L)):
Time complexity:-
where L is the length of word that is searched for in a constant time but its not O(1) because its cost depends on the length of the word that is searched for. It might vary from one word to another. but doesn't depends on the number of nodes in this Try

Note:
In Java when alphabet is subtracted from the characters it is equal to the index where you want to store the value of your target alphabet. this is because the alphabets are stored in a numeric value in the memory

↳ Insert (O(L))
Time complexity:
One has to go down and search or traverse through the tree if character doesn't exists it is added into it. Number of operations involved is equal to no. of characters that is being inserted.

5 FRIDAY الجمعة ٥

7 Rabi' II 1427H | ٧ ربيع الثاني ١٤٢٧

↳ Deletion (O(L))
Time Complexity.
Works same as the insertion

↳ Insert:-
In insertion certain nodes are marked current in order to add more or traverse through the nodes for future.

NOTES ملاحظات

JANUARY يناير/كانون الثاني
S S M T W T F
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

FEBRUARY فبراير/شباط
S S M T W T F
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28

MARCH مارس/آذار
S S M T W T F
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31

APRIL أبريل/نيسان
S S M T W T F
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

MAY مايو/أيار
S S M T W T F
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

JUNE يونيو/حزيران
S S M T W T F
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

JULY يوليو/تموز
S S M T W T F
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

AUGUST أغسطس/آب
S S M T W T F
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

SEPTEMBER سبتمبر/أيلول
S S M T W T F
30 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29

OCTOBER أكتوبر/تشرين الأول
S S M T W T F
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

NOVEMBER نوفمبر/تشرين الثاني
S S M T W T F
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

DECEMBER ديسمبر/كانون الأول
S S M T W T F
30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29

MAY مايو/أيار 2006/ ٢٠٠٦

6 SATURDAY السبت ٦ | 7 SUNDAY الأحد ٧ | 8 MONDAY الاثنين ٨

8 Rabi' II 1427H ٨ ربيع الثاني ١٤٢٧هـ | 9 Rabi' II 1427H ٩ ربيع الثاني ١٤٢٧هـ | 10 Rabi' II 1427H ١٠ ربيع الثاني ١٤٢٧هـ

↳ Abstraction:

↳ The inner working of the classes should not be exposed to the outside one. in a program.

↳ for example think of a remote control. the logic board is complex and can be understood by the user so the developer give's some button to the user and let them operate with it. So that he/she doesn't need to understand the logistics working behind pressing those button.

↳ Same thing goes with the programming.

NOTES ملاحظات | NOTES ملاحظات | NOTES ملاحظات

9 TUESDAY الثلاثاء ٩ | 10 WEDNESDAY الأربعاء ١٠ | 11 THURSDAY الخميس ١١

11 Rabi' II 1427H ١١ ربيع الثاني ١٤٢٧هـ | 12 Rabi' II 1427H ١٢ ربيع الثاني ١٤٢٧هـ | 13 Rabi' II 1427H ١٣ ربيع الثاني ١٤٢٧هـ

(11) Graphs :-

↳ Socials networks, GPS or where relationship between objects are modelled.

↳ Connected objects.

↳ Consists of nodes edges just like a tree but without a root node.

↳ Nodes in graphs are called 'vertices' or singular as 'vertex'.

12 FRIDAY الجمعة ١٢

14 Rabi' II 1427H ١٤ ربيع الثاني ١٤٢٧هـ

↳ That's why these nodes can be connected to many other nodes. So there is no limitation to it.

↳ If two nodes directly connected they are adjacent/~~nodes~~ neighbour.

↳ If nodes have direction thats called a directed graph. Just like Twitter.

↳ Facebook and LinkedIn are undirected graphs as it will show connections to others to like friend suggestions.

↳ Connections in graphs has weights which shows how strong that connection is.

NOTES ملاحظات | NOTES ملاحظات

JANUARY يناير/كانون الثاني | FEBRUARY فبراير/شباط | MARCH مارس/آذار | APRIL أبريل/نيسان | MAY مايو/أيار | JUNE يونيو/حزيران

JULY يوليو/تموز | AUGUST أغسطس/آب | SEPTEMBER سبتمبر/أيلول | OCTOBER أكتوبر/تشرين الأول | NOVEMBER نوفمبر/تشرين الثاني | DECEMBER ديسمبر/كانون الأول

3 SATURDAY السبت ١٣ | 14 SUNDAY الأحد ١٤ | 15 MONDAY الاثنين ١٥

↳ Adjacency Matrix :-

↳ It requires a two dimensional array or a matrix with rows or columns.

↳ But it requires double the amount of space to store in the memory like for 'n' nodes, 'n²' space is required. like space complexity for it will be:

Space : $O(n^2)$ or $O(V^2)$

↳ Operations with ~~runtime~~ complexity :-

↳ Add node : $O(n^2)$ / space

But when talking about graphs we take vertices 'V' or edges 'E' in order to calculate it in graphs instead of 'n'. As

Add node: $O(V^2)$ as $n = V$

↳ Remove node : $O(V^2)$

NOTES ملاحظات

↳ Add edge : $O(1)$

↳ Remove edge : $O(1)$

↳ Query edge : $O(1)$

↳ Find neighbours : $O(V)$

16 TUESDAY الثلاثاء ١٦ | 17 WEDNESDAY الأربعاء ١٧ | 18 THURSDAY الخميس ١٨

↳ Adjacency list :-

↳ It requires array of linked lists. This linked list contains adjacent node or neighbours of a given node.

↳ So instead of matrix only the edges that exist in a graph are stored in these lists.

↳ That's it more space efficient than the matrix.

19 FRIDAY الجمعة ١٩

↳ Operations with ~~runtime~~ / space Complexity

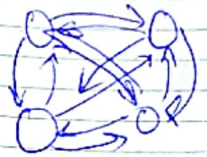
Space : $O(V)$

But E = total edges so adjacency list has a space complexity of

Space : $O(V + E)$

Dense graph:

↳ When every node is connected to every other node.



$E = V \times (V - 1)$ E = total edges, V = vertices

or $E = V^2 - V$ or

space complexity $O(V + E) = O(V^2)$

↳ Operations with time-complexity:-
  ↳ Add node: O(1)
  ↳ Remove node: O(V) but when comes the 'E' edges then it will be O(V+E) as these edges will be accross all the graph connecting 'V' nodes.

But when comes to the dense graph the worst case which is to be considered is with

$E \simeq V^2$ so $O(V^2)$

↳ If 'K' is number of edges of a given node then
  ↳ Add edge: O(K)
    worst case: O(V)
  ↳ In multigraphs two nodes were connected with multiple edges, So, in that case
    Add edge: O(1)
  ↳ Remove edge: O(K)
    worst case: O(V)
  ↳ Query edge: O(K)
    worst case: O(V)

↳ Find neighbours: O(K)
  worst case: O(V)
  ↳ So if graph is not dense this above graph is gonna run much faster, as one doesn't need to check every other node.

	Matrix	List
↳ Space	$O(V^2)$	$O(V+E)$
Add edge	O(1)	O(K)
Remove edge	O(1)	O(K)
Query edge	O(1)	O(K)
Find neighbours	O(V)	O(K)
Add node	$O(V^2)$	O(1)
Remove node	$O(V^2)$	$O(V^2)$

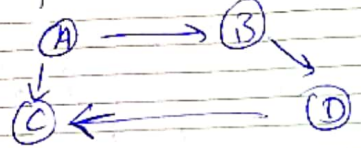
↳ In graphs we can traverse from any node as root node that nodes or vertices are directly or indirectly connected to each other.
  ↳ Breadth first, depth first traversal.
    using queues to store values as going in a tree.

MAY مايو/أيار — 2006/ ٢٠٠٦

27 SATURDAY السبت — ٢٧ — 28 SUNDAY الأحد — ٢٨ — 29 MONDAY الاثنين — ٢٩

29 Rabi' II 1427H — ٢٩ ربيع الثاني ١٤٢٧هـ — 1 Jumada I 1427H — ١ جمادى الأول ١٤٢٧هـ — 2 Jumada I 1427H — ٢ جمادى الأول ١٤٢٧هـ

↳ Depth - first traversal :-
↳ It is done using recursion function



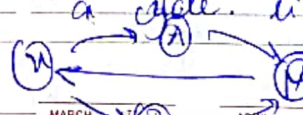
↳ Values are stored in sets, while traversing in graph using depth first traversal technique.

↳ Breadth - first traversal.-
↳ Queues are used instead of stacks, sets while visiting the nodes.

↳ Topological Sorting:-
↳ Right order for processing the nodes of the graph
↳ like scheduling jobs

In different projects different jobs are done simultaneously thats known as topological sorting.

Directed Acyclic graph (DAG) → ↳ It only works in graphs without a cycle. like



↳ in order to build X P is built first ↳ but for order to build P X built first

↳ Topological doesn't work if there is a cycle

JUNE يونيو/حزيران — 2006/ ٢٠٠٦

30 TUESDAY الثلاثاء — ٣٠ — 31 WEDNESDAY الأربعاء — ٣١ — 1 THURSDAY الخميس — ١

3 Jumada I 1427H — ٣ جمادى الأول ١٤٢٧هـ — 4 Jumada I 1427H — ٤ جمادى الأول ١٤٢٧هـ — 5 Jumada I 1427H — ٥ جمادى الأول ١٤٢٧هـ

↳ Cycle detections.
In depth first
↳ By dividing nodes in three categories
↳ All nodes
↳ Visiting nodes
↳ Visited nodes

In Breadth first ↳ Through hash tables by dividing nodes and parent nodes

2 FRIDAY الجمعة — ٢

6 Jumada I 1427H — ٦ جمادى الأول ١٤٢٧هـ

(2) Undirected Graphs
↳ Edges don't have direction
↳ Edges that have weights are called weighted graphs such weights are costs, distance, complexity or anything.

~~↳ Digits~~

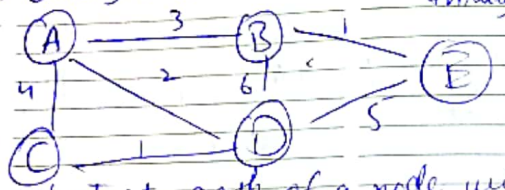
3 SATURDAY السبت ٣ 4 SUNDAY الأحد ٤ 5 MONDAY الاثنين ٥

7 Jumada'l 1427H | ٧ جمادى الأول ١٤٢٧هـ | 8 Jumada'l 1427H | ٨ جمادى الأول ١٤٢٧هـ | 9 Jumada'l 1427H | ٩ جمادى الأول ١٤٢٧هـ

↳ Dijkstra's shortest path algorithm:-

↳ Finding shortest path between two nodes of a tree/graph.

↳ Greedy algorithm: making optimal solutions thraugh optimal choices at every step



↳ Shortest path of a node with current neighbour

Node	Distane	Previous
A	0	
B	~~Max~~ 3	A
C	Max, 4	A
D	Max, 2	A
E	Max, 7	A

Shortest path ←

↳ Shortest path from one target node to the other node which has shortest paths for all other nodes.

↳ like from A to E via D is 2+5 = 7

↳ And so on with all other nodes until a shortest path is known.

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL ابريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

6 TUESDAY الثلاثاء ٦ 7 WEDNESDAY الأربعاء ٧ 8 THURSDAY الخميس ٨

10 Jumada'l 1427H | ١٠ جمادى الأول ١٤٢٧هـ | 11 Jumada'l 1427H | ١١ جمادى الأول ١٤٢٧هـ | 12 Jumada'l 1427H | ١٢ جمادى الأول ١٤٢٧هـ

↳ values can be stored in hash tables:

Map <Node, integer> distances

Map <Node, Node> previous Nodes.

↳ Minimum spanning tree:-

↳ Converted a graph into a tree.

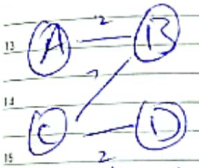
↳ A tree is without a cycle

↳ Reducing the number of edges in order to reduce the cost that connects nodes to other nodes.

↳ So E = N-1 if 'N' is nodes in a graph

↳ Cost = weight of the edges.

↳ Minimum cost results in efficiency of the tree.



total cost = 2+2+2 = 11

9 FRIDAY الجمعة ٩

13 Jumada'l 1427H | ١٣ جمادى الأول ١٤٢٧هـ

↳ Prim's Algorithm:-

↳ Extend the tree by adding the smallest connected edges.

↳ Another type of greedy algo.

NOTES ملاحظات

JULY يوليو/تموز

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST اغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					
2	3	4	5	6	7	
9	10	11	12	13	14	
16	17	18	19	20	21	
23	24	25	26	27	28	

↳ Implementing prim's :-
↳ Using priority Queue
↳ Repeat until tree has all the nodes
↳ Represent the tree using weighted graph.

NOTES ملاحظات

(13) Sorting Algorithms:-
↳ Bubble sort:-
↳ Items are sorted in ascending order.
↳ But this sorting takes place in a sequence. This sorting is done through swapping the elements in an array like with smaller and bigger values until the result is sorted out in an ascending order.

↳ Time complexity

	Best	Worst
Passes	O(1)	O(n)
Comparisons	O(n)	O(n) or O(n-1)
Total	O(n)	$O(n^2)$
	Linear	Quadratic

↳ We take one value and compare to all others until the array is fully sorted out.

NOTES ملاحظات

17 SATURDAY السبت ١٧ | 18 SUNDAY الأحد ١٨ | 19 MONDAY الاثنين ١٩

21 Jumada' I 1427H | 22 Jumada' I 1427H | 23 Jumada' I 1427H

↳ Selection Sorting:-

↳ Multiple passes are required to sort out the array.

↳ With each passe next smallest array is found out and moved to the correct position.

↳ It has two parts, the sorted one and the next is unsorted one.

↳ Time Complexity:

	Best	Worse
Passes	O(n)	O(n)
Comparisons	O(n)	O(n)
Total	$O(n^2)$	$O(n^2)$
	Quadratic	Quadratic

∵ Determines that its slower algorithm

NOTES ملاحظات

↳ It is better than bubble because in one pass we come to know that either sorted or not. To stop further passes or continue. But in bubble we did comparisons until it sorted out.

↳ We take one value and when it's position we move one to next

20 TUESDAY الثلاثاء ٢٠ | 21 WEDNESDAY الأربعاء ٢١ | 22 THURSDAY الخميس ٢٢

24 Jumada' I 1427H | 25 Jumada' I 1427H | 26 Jumada' I 1427H

↳ Insertion Sorting :-

↳ Works just like a card game.

↳ We get the value / item we store it and move on to next value in array and put them or store them back in array in correct position.

↳ Bigger values items are shifted to the right in open space so that the value of of small items are store on that place which are taken for a temporary storing point.

23 FRIDAY الجمعة ٢٣

27 Jumada' I 1427H

↳ Time Complexity:

	Best	Worse
Iteration	O(n)	O(n)
Shift item	O(1) if array sorted	O(n) if array is unsorted/descending order
Total	O(n)	$O(n^2)$
	linear	Quadratic.

NOTES ملاحظات

↳ Merge Sorting:-

↳ Breakdown a list into smaller and smaller sublists, sort those and merge them back to a completely sorted list.

↳ It is done by dividing an array into half by;

middle = length of array / 2

↳ An array is split into two sub-arrays, allocate two smaller arrays, copy items/elements from an input / additional array into these two allocated arrays.

↳ This takes more space in the memory.

↳ The two splited arrays are compared then and values are allocated into the merged one or new array.

↳ Merge sort also known as "Divide and Conquer."

NOTES ملاحظات

↳ Space Complexity:-

	Best	worse
Space	O(n)	O(n)

↳ Time complexity:-

	Best	Worst
Dividing	O(log n)	O(log n)
Merging	O(n)	O(n)
total	O(n log n)	O(n log n)

~ Merge sort is faster than its predecessors but acquires additional space.

↳ Quick Sorting:-

↳ Mostly used in different frameworks.

↳ Friendly, efficient, doesn't require any additional space.

↳ It starts with selecting an item named as pivot.

↳ Items that are smaller than the pivot are on the left and those which are greater than the pivot are on the right side of it also known as partitioning.

↳ Typically last item is selected as the pivot but it varies.

↳ Two pointers are required one to iterate over an array and other one to mark the boundry the array while doing partitioning.

↳ Boundary elements or elements/items coming in boundry pointer are not touched while the other pointer is iterating over the items and swaps smaller items with the items in the boundry if they are bigger.

↳ An array can be divided into half: best. O(log n) if pivot is in the middle.

↳ But if pivot is last item or it is a sorted array in (asscending order) it will worse to partition an array O(n)

↳ works in worse this manner when pivot is first i.e (descending order)

JULY يوليو/تموز 2006/ ٢٠٠٦

1 SATURDAY السبت — 12 SUNDAY الأحد — 23 MONDAY الاثنين — ٣

5 Jumada II 1427H — ٥ جمادى الثاني ١٤٢٧هـ — 6 Jumada II 1427H — ٦ جمادى الثاني ١٤٢٧هـ — 7 Jumada II 1427H — ٧ جمادى الثاني ١٤٢٧هـ

↳ Pivot Selection:
- ↳ Pick randomly
- ↳ Use the middle index
- ↳ Average of first, middle and last item.

↳ Time complexity:-

	Best	Worse
Partition	$O(n)$	$O(n)$
no. (#) of Time an array partitioned	$O(\log n)$	$O(n)$
Total	$O(n \log n)$	$O(n^2)$

∴ with good Pivot selection it runs on $O(n \log n)$ and becomes the efficient algorithm.

NOTES ملاحظات

Space

NOTES ملاحظات

$O(\log n)$

∵ But it takes additional space due to recursive calls stack.

NOTES ملاحظات

$O(n)$

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

JULY يوليو/تموز 2006/ ٢٠٠٦

4 TUESDAY الثلاثاء — 45 WEDNESDAY الأربعاء — 56 THURSDAY الخميس — ٦

8 Jumada I 1427H — ٨ جمادى الثاني ١٤٢٧هـ — 9 Jumada II 1427H — ٩ جمادى الثاني ١٤٢٧هـ — 10 Jumada II 1427H — ١٠ جمادى الثاني ١٤٢٧هـ

↳ Counting Sort
- ↳ Non-comparison sort algorithm but uses basic math
- ↳ figures how many times an item appears in an array.
- ↳ Those repeating counts or frequencies are stored in another array.
- ↳ Through later we sort out which item should come first and which one should come last in an array.

7 FRIDAY الجمعة — ٧

11 Jumada II 1427H — ١١ جمادى الثاني ١٤٢٧هـ

↳ Instead of Comparisons we use the counting of the occurances of the items in an input array, and sort the array.

↳ Time - Space Complexity:-

Space: $O(K)$ if 'K' is the max value in the input array.
- ↳ More space is allocated for counts array. Size of this array depends on max value of the input array.

Time:

populate counts $O(n)$ ⇒ while iterating over an input array.

Iterate counts $O(K)$ ⇒ 'K' is max value

Total $O(n+K)$ or $O(n)$ linear (much faster than predecessors)

NOTES ملاحظات

NOTES ملاحظات

NOTES ملاحظات

JULY يوليو/تموز

S	S	M	T	W	T	F
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

↳ Time Memory trade-off

↳ when algorithms or programs are made run faster at the cost of extra memory.

↳ So there is (execution) trade-off between execution time and memory

↳ Counting sort requires an extra space but runs faster so it's the perfect example of time memory trade-off

↳ Counting Sort is used when:

↳ Allocating extra space is not an issue

↳ Values should be positive integers otherwise we can't use them as indexes of the count's array.

↳ Best if most of the values in the range are present in the input array, otherwise one will end up with lots of empty or zero elements in count's array

↳ Counting sort is suitable when the most values in the range are present

↳ Bucket Sort

↳ Doesn't use comparisons

↳ Distribute items in a number of buckets, sort that bucket by using another sorting algorithm and then combine them.

↳ In this way instead of sorting large arrays we sort buckets and this takes way less time

14 FRIDAY الجمعة ١٤

↳ Interestingly these buckets can be sorted in parallel.

↳ For finding the current bucket:-

bucket = item / no. of buckets

↳ The more the buckets the more the memory it will take but can sort each bucket in fewer times, due to fewer items in each bucket

NOTES ملاحظات

↳ Space-Time Complexity:-

Space : O (n + K) - K is the no. of buckets and ~~items~~ ~~no.~~ K = items

Time	Best	Worst
Distribution	O(n)	O(n)
Iterating buckets	O(K)	O(K)
Sorting	O(1)	O(n²)
Total	O(n+K)	O(n²) → fewer buckets or O(n) → more buckets with more memory cost

Scanned with CamScanner

15 SATURDAY السبت ١٥ | 16 SUNDAY الأحد ١٦ | 17 MONDAY الاثنين ١٧

19 Jumada II 1427H | 20 Jumada II 1427H | 21 Jumada II 1427H

(14) Searching algorithms:-

~~↳ Simplest~~

↳ Linear Search

↳ Simplest algorithm

↳ Iterate over a list inspect each item, if found out that item, return its index otherwise -1, means that item doesn't exist.

↳ Time Complexity

Best	Worst
O(1) compare when at the beginning of list	O(n) compare when at the end of list or doesn't exist at all.

Target item

↳ This algo is slow when comes to large lists but fine with small lists.

NOTES ملاحظات

18 TUESDAY الثلاثاء ١٨ | 19 WEDNESDAY الأربعاء ١٩ | 20 THURSDAY الخميس ٢٠

22 Jumada II 1427H | 23 Jumada II 1427H | 24 Jumada II 1427H

↳ Binary Search

↳ Faster than linear search but works with sorted lists only.

↳ Calculating index of middle item:

middle index = (left + right / 2)

where left, right are index of items

↳ We cut down the number of items that we are inspecting through above formula.

21 FRIDAY الجمعة ٢١ | 25 Jumada II 1427H

↳ Just like "Divide and Conquer"

↳ Time Complexity:-

O(log n) Because

Target value == mid

target > mid

target < mid

↳ Space Complexity:-

O(log n) ∴ when use recursion

O(1) ∴ when use iteration.

NOTES ملاحظات

JANUARY يناير/كانون الثاني

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY فبراير/شباط

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH مارس/آذار

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL أبريل/نيسان

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY مايو/أيار

S	S	M	T	W	T	F
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE يونيو/حزيران

S	S	M	T	W	T	F
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

JULY يوليو/تموز

S	S	M	T	W	T	F
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST أغسطس/آب

S	S	M	T	W	T	F
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER سبتمبر/أيلول

S	S	M	T	W	T	F
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER أكتوبر/تشرين الأول

S	S	M	T	W	T	F
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER نوفمبر/تشرين الثاني

S	S	M	T	W	T	F
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER ديسمبر/كانون الأول

S	S	M	T	W	T	F
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

↳ Ternary Search:

↳ the list is divided into three parts.

↳ Calculating size of partition:-

partition size = array.length/3

but only at the beginning so for that we use.

partition size = (right − left)/3 "right, left are indexes" then

mid 1 = left index + partitionsize

mid 2 = right index − partitionsize

↳ Time Complexity:-

$O(\log_3 n)$

Because,

5 comparisons in every step.

target value > mid2
target == mid2
target < mid2 && target > mid1
target == mid1
target < mid1

In worst case scenario the number of comparisons to find a number is more in ternary search

Note:

↳ Logarithm helps us find the power in

$2^x = 8$

So, $\log_2 8 = 3$

↳ Also tells us to which power should one number be raised to another number.

Note: Binary search is faster than ternary

↳ Jump Search:-

↳ Improved version of linear search but not as fast as binary search.

↳ With this search algorithm we divide the list into few blocks and instead of checking every item, we jump to the block where target item may exist.

↳ then linear search is performed on only that block.

↳ That is why it's a jump search because it jumps to particular block.

↳ Calculating block size:-

blocksize = $\sqrt{n}$: where 'n' is number of items in an array

↳ Time Complexity:-

$O(\sqrt{n})$

↳ Two pointers are required, first to determine the start of 1st block and 2nd for the start of next block.

↳ Edge cases (jumps out of boundry)

↳ start >= length of array

↳ To avoid crash of algorith one should consider that it doesn't exceed length of array

↳ Exponential Search :-

↳ Start from small range and check whether ideal item is in that range or not.

↳ Otherwise range will be be doubled with every step, range is the search of a specific item in a specific number of items. After that a binary search is implemented.

↳ A 'bound' variable is declared which stores the index of last item in a range.

↳ lower the index no. ~~value~~ then it is lower bound.

↳ higher the index no. ~~value~~ then it is ~~higher~~ upper bound.

↳ Time Complexity

Worst:

$O(\log i)$

∴ Binary search over an array of 'i' elements.

NOTES ملاحظات

↳ When the value we are looking for is greater than the previous bound then to find that value between the bound which has greater value than the previous bound ~~having~~ value smaller than target value we use: range: [bound/2, bound]

↳ Now binary search is done on this range.