National University of Colombia
Algorithms
(Group 1)

Lab4

Cristian David Tafur Devia

October 2018

1 Complete the following table

Algorithm	Function	Worst case time complexity	Best case time complexity	Average case time complexity	Space complexity
The simplest primality by trial division	Dado un numero de entrada n, verifica si algún entero primo m de 2 a n divide uniformemente n sin dejar resto. Si n es divisible por cualquier m, entonces n es compuesto, de lo contrario es primo.	O($\sqrt{n}$)	O($\sqrt{n}$)	O($\sqrt{n}$)	O($\sqrt{n}$)
Binary search	Algoritmo de búsqueda que encuentra la posición de un valor en un array ordenado. Compara el valor con el elemento en el medio del array, si no son iguales, la mitad en la cual el valor no puede estar es eliminada y la búsqueda continúa en la mitad restante hasta que el valor se encuentre.	O(log(n))	O(1)	O(log(n))	O(1)
Finding the smallest or largest item in an unsorted array	1) Almacene los primeros k elementos en una matriz temporal temp [0..k-1]. 2) Encuentra el elemento mas pequeño / mas grande en temp [] 3) Para cada elemento x en arr [k] hasta arr [n 1] Si x es mayor que el numero, lo elimina de tem[] e inserte x. 4) Imprime los elementos de temp	O((n-k)*k)	O((n-k)*k)	O((n-k)*k)	6

Algorithm	Function	Worst case time complexity	Best case time complexity	Average case time complexity	Space complexity
Kadanes Algotirhm	Busca todos los segmentos contiguos positivos de la matriz. Y realiza un seguimiento de la suma maxima del segmento contiguo entre todos los segmentos positivos. Cada vez que obtiene una suma positiva, la compara con max y la intercambia si es mayor	O(n)	O(n)	O(n)	O(1)
Sieve of Eratosthenes	Algoritmo que permite hallar todos los numeros primos menores que un numero natural dado n	O(n log log(n))	O(n log log(n))	O(n log log(n))	O(n)
Merge Sort	Divide el arreglo de llegada en dos partes, repitiendo este proceso para cada segmento dividido, ordenando el arreglo más pequeño para luego mezclarlo con los demás ordenando por completo el arreglo final	O(n log(n))	$\Omega(nlog(n))$	$\theta(nlog(n))$	O(n)
Heap Sort	Encuentra el máximo elemento y lo ubica en el final, repitiendo el proceso para cada elemento restante	O(log(n))	$\Omega(nlog(n))$	$\theta(nlog(n))$	O(1)
Quick Sort	El destino de las particiones es, dada una matriz y un elemento x de la matriz como pivote, coloca x en su posicion correcta en la matriz ordenada y coloca todos los elementos mas pequeños (mas peque˜nos que x) antes de x, y coloca todos los elementos mayores (mayores que x) después de X	$O(n^2)$	$\Omega(nlog(n))$	$\theta(nlog(n))$	O(log(n))
Tim Sort	Dividimos el Array en bloques conocidos como Run. Ordenamos esas ejecuciones utilizando insertion sort una por una y luego las combinamos con la funcion de combinacion utilizada en merge sort. Si el tama˜no de Array es menor que el de ejecucion, entonces Array se ordena simplemente utilizando Insertion Sort. El tama˜no de ejecucion puede variar de 32 a 64 dependiendo del tama˜no de la matriz. Tenga en cuenta que la funcion merge funciona bien cuando los arreglos secundarios de tama˜no son potencias de 2. La idea se basa en el hecho de que insertion sort funciona bien para arreglos peque˜nos.	O(n log(n))	$\Omega(n)$	$\theta(nlog(n))$	O(n)

Algorithm	Function	Worst case time complexity	Best case time complexity	Average case time complexity	Space complexity
Convex Hull using Divide and Conquer Algorith	Parte de la suposicion del conocimiento previo del casco convexo de la mitad izquierda y derecha, para fusionarlos, se encuentra la tangente superior e inferior a los cascos convexos derecho e izquierdo	O(nlog(n))	O(nlog(n))	O(nlog(n))	
Insertion Sort	Toma el elemento i+1 del arreglo y lo compara con toda la lista previamente ordenada, deteniendose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posicion a la derecha)o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el mas peque˜no). En este punto se inserta el elemento i+1 desplazando los demas elementos.	$O(n^2)$	O(n)	$O(n^2)$	O(1)
Floyd–Warshall algorithm	Se inicializan las matrices Se consideran vertices como intermedios y se actualiza la matriz Se eligen todos los vertices y se actualizan todas las rutas mas cortas como un vertice intermedio en la ruta mas corta. Para cada (i, j) de los vertices de origen y destino, hay dos casos posibles. 1) k no es un vertice intermedio en la ruta mas corta de i a j. Mantenemos el valor de dist [i][j] tal como es. 2) k es un vertice intermedio en la ruta mas corta de i a j. Actualizamos el valor de dist [i][j] como dist [i][k] + dist [k] [j] si dist [i][j] ¿dist [i][k] + dist [k][j]	$\theta(\|V\|^3)$	$\theta(\|V\|^3)$	$\theta(\|V\|^3)$	$\theta(\|V\|^2)$
Calculate the permutations of n distinct elements without repetitions	Las permutaciones sin repeticion de n elementos son los distintos grupos de n elementos que se pueden hacer, de forma que dos grupos se diferencian unicamente en el orden de colocacion de los elementos. Consideremos el conjunto A = a, b, c, d, e Entonces las permutaciones de estos elementos son: abcde, acbde, dbeca, adcea, bedac, cdbae, caebd, edabc, etc.	$O(n^2 * n!)$			

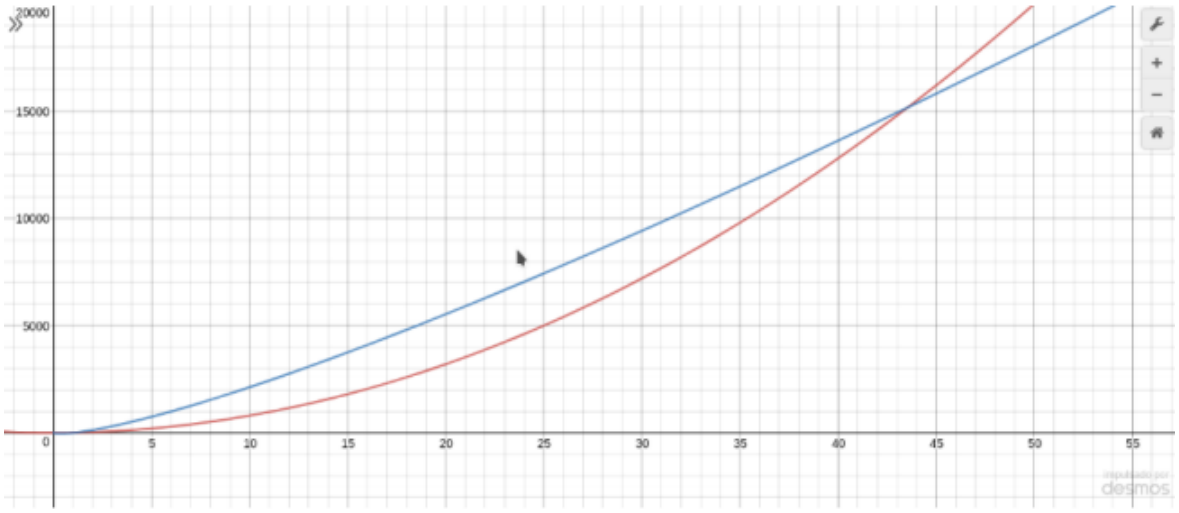
Algorithm	Function	Worst case time complexity	Best case time complexity	Average case time complexity	Space complexity
Dijkstra's algorithm	Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial. Un vector D de tama˜no N guardara al final del algoritmo las distancias desde x hasta el resto de los nodos. 1. Inicializar todas las distancias en D con un valor infinito, exceptuand o la de x, que se debe colocar en 0. 2. Sea a = x (a como nodo actual.) 3.Se recorren todos los nodos adyacentes de a, excepto los nodos marcados. Se les llamara nodos no marcados vi. 4. Para el nodo actual, se calcula la distancia tentativa desde dicho nodo hasta sus vecinos con: dt(vi) = Da + d(a,vi).Esdecir, la distancia tentativa de 'vi0 es la distancia que actualmente tiene el nodo en el vector D mas la distancia desde dicho nodo 'a 0 hasta el nodo vi. Si la distancia tentativa es menor que la distancia almacenada en el vector, se actualiza el vector con esta distancia tentativa. Es decir, si dt(vi) ¡ DviDvi = dt(vi) 5. Se marca como completo el nodo a. 6. Se toma como proximo nodo actual el de menor valor en D y se regresa al paso 3, mientras existan nodos no marcados. Una vez terminado al algoritmo, D estara completamente lleno.	O($\mid E\mid + \mid V\mid log\mid V\mid$)	O($\mid V\mid^2$)		O($\mid V\mid+ \mid E\mid$)
Naive Matrix Inversion	Si A es una matriz cuadrada n n, entonces se puede usar la reduccion de fila para calcular su matriz inversa, si existe. Primero, la matriz de identidad n n se aumenta a la derecha de A, formando una matriz de bloque n2n[A—I]. Ahora a travesde la aplicacion de operaciones de fila elementales, encuentre la forma escalonada reducida de esta matriz n2n. La matriz A es invertible si y solo si el bloque izquierdo puede reducirse a la matriz de identidad I; en este caso, el bloque derecho de la matriz final es A 1. Si el algoritmo no puede reducir el bloque izquierdo a I, A no es invertible.	O(n^3)	O(n^3)	O(n^3)	

Algorithm	Function	Worst case time complexity	Best case time complexity	Average case time complexity	Space complexity
Calculate the permutations of n distinct of n distinct with repetitions	Las permutaciones con repeticion de n elementos en las que el primer elemento se repite n1veces, el segundo n2 veces, ... y el ultimo se repite nreces, son los distintos grupos de n elementos que se pueden hacer de forma que en cada grupo, cada elemento aparezca el numero de veces indicado. Ademas, dos grupos se diferencian unicamente en el orden de la colocacion. Se representa por Pnn1,..,nr. Para saber cuantas permutaciones con repeticion de n elementos, en las que el primer elemento se repite n1veces, el segundo n2veces, ... y el ultimo se repite n veces.	O(n*m)			O(n)

2 Cormen Exercises

2.1 Exercise 1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in 64 n lg n steps. For which values of n does insertion sort beat merge sort?

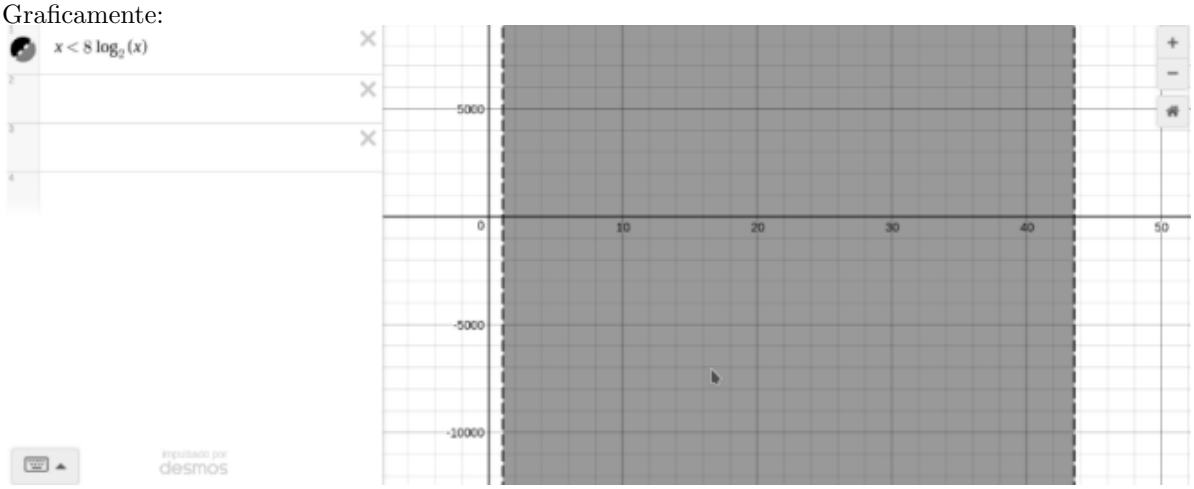


$$t = 8n^2, t = 64nlog(n)$$

Se realiza la siguiente desigualdad:

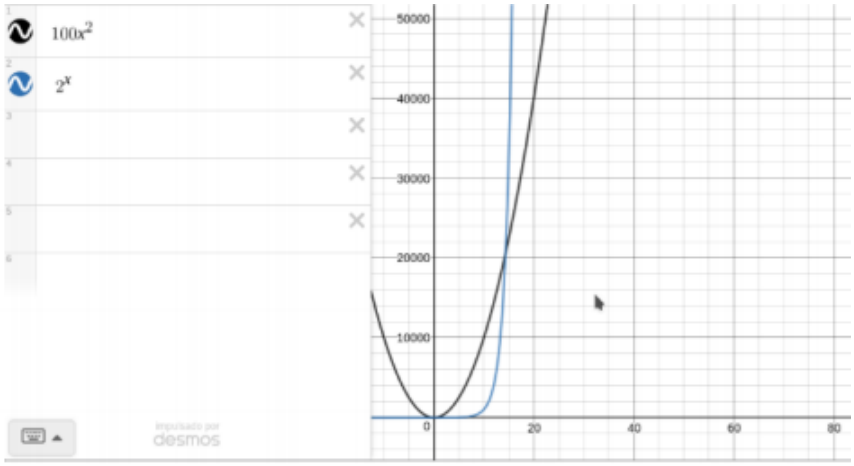
$$8n^2 < 64nlog(n), paralocual : n < 8log(n)$$

Graficamente:



$$(1.1 < n < 43.55993)$$
$$(1 < n < 43)$$

2.2 Exercise 1.2-3

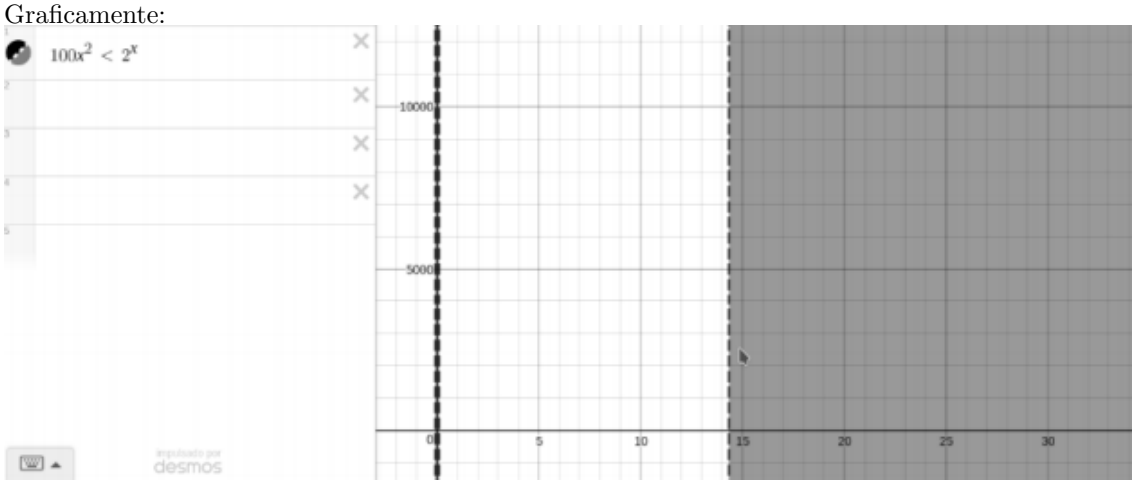


$$t = 100n^2$$
$$t = 2^n$$

Desigualdad:

$$100n^2 < 2^n$$

Graficamente:



$$(n > 14.32)$$

Respuesta: 15

2.3 Problem 1-1 - solve from 1 microsecond (10^-6s) for step to for 1 nanoseconds (10^-9s) for step

	Equivalencia	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
microseconds		1e6	6e7	3.6e9	8.64e10	2.628e12	3.154e13	3.156e15
lg n	n = 2^t	2^(e6)	2^(6e7)	2^(3.6e19)	2^(8.64e10)	2^(2.628e12)	2^(3.153e13)	3.156e15
√n	n = t^2	1e12	3.6315	1.29e19	7.46e21	7.72e24	9.95e26	9.96e30
n	n = t	1e6	6e7	3.6e9	8.64e10	2.628e12	3.154e13	3.156e15
n lg n	n = t*log10(t)/log10(2)	62764	2801417	133378058	2755147513	71870856404	797633893349	6.86e13
n^2	n = √t	1000	7745	60000	293938	1609968	5615692	56176151
n^3	n = 3√t	100	391	1532	4420	13736	31593	146679
2^n	log 2^n = log2 t	19	25	31	36	41	44	51
n!	...	9	11	12	13	15	16	17

2.4 Problem 3-1

a)

$$polinomiop(n) -> f(n)$$
$$O(n^k) -> g(n)$$
$$0 <= f(n) <= cg(n) paratodon >= 0$$
$$cg(n) - f(n) >= 0$$
$$cg(n) >= f(n)$$
$$R/n^k >= p(n)$$

b)

$$p(n) -> f(n)$$
$$n^k -> g(n)$$
$$0 < cg(n) <= f(n) paratodon >= 0$$
$$f(n) - cg(n) >= 0$$
$$f(n) >= cg(n)$$

Respuesta:

$$p(n) >= n^k$$

c) Por el siguiente Teorema
Teorema
For any two functions $f(n)$ and $g(n)$, we have $f(n) = \theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Como n^k es cota superior de p(n) O y cota inferior Ω entonces también es $\theta(n^k)$

3 Dasgupta, Papadimitriou and Vazirani

3.1 Exercise 0.1

En cada una de las siguientes situaciones, indique si f = O (g), o $f = \Omega(g)$, o ambos (en cuyo caso $f = \theta(g)$).
Tenemos que:
f = O(g(n)) significa que f(n) no crece tan rapido como g(n)
f = $\Omega(g(n))$ significa que g = O(f(n))
f = $\theta(g(n))$) indica que f(n)=O(g(n)) y g(n)=O(f(n)) o ($f = \omega(g(n))$) lo que significa que f = O(g(n)) y $f = \Omega(g(n))$

Para cada uno de los items, se graficaron las funciones propuestas para comparar si eran cotas superiores, inferiores o ambas en dado caso.

3.2 Exercise 0.2

Fórmula de serie geométrica:

$$\sum_{i=1}^{n} ar(k-1) = \ a \bullet \frac{(1-r^n)}{(1-r)}$$

Entonces:

$$g(n) = \frac{c \bullet (n+1) - 1}{c-1}$$

a) $c < 1$

$$\lim_{x \to \infty} \frac{(0-1)}{(c-1)} = \frac{1}{1-c}$$

$$1 > 1 - cn + 1 > 1 - c$$

$$\frac{1}{1-c} > g(n) > 1$$

Como el valor de los términos está disminuyendo, se puede determinar que esta operación es $\theta(1)$

$b) c = 1$
$g(n) = 1 + 1 + 1 + ... + 1 = n + 1 = O(n)$
c) $c > 1$

$$cn + 1 > cn + 1 - 1 > cn$$

$$\frac{c}{1-c} \bullet cn > g(n) > \frac{1}{1-c} \bullet cn$$

Como el valor de los términos está en aumento, se puede concluir que esta operación es $\theta(cn)$

3.3 Solve T(n) = 2 T(n-2) + 2

$$T(n) = 2T(n-2) + 2, n = 2k$$

$$= 2(2T(n-4) + 2) + 2$$

$$= 4T(n-4) + 2 * 2 + 2$$

$$= 4(2T(n-6) + 2) + 2 * 2 + 2$$

$$= 2^3 T(n - 2 * 3) + 2^3 + 2^2 + 2$$

$$= 2^k T(0) + 2^(k-1) + 2^(k-2) + ... + 2^2 + 2$$

$$= 2^k - 1 - 1$$

$$= 2^k - 2 = 2^(\frac{n}{2}) - 2$$