Facultad Politécnica – Ingeniería Informática
Algoritmos y Estructura de Datos III – 2022 – 1er. Periodo – Sección TQ/TR

Tarea 5

Trabajo a entregar por GRUPO vía EDUCA. Indique claramente los integrantes de su grupo. El código debe poder probarse sin usar ningún IDE desde la linea de comandos. No incluya ninguna clase en package.

Parte Práctica (35p)

Ejercicio 1 (10p)

Escriba un programa Java que lea N puntos en un plano (puede dejar los datos puestos en la función main) e imprima cualquier grupo de cuatro o más puntos colineales (ésto es, puntos en la misma línea). Implemente un algoritmo para este problema y justifique su tiempo de cómputo.

Levantar en VPL el código y colocar la fundamentación del tiempo como comentario en la actividad.

Observación: Se debe observar la siguiente rúbrica:

Rúbrica	Puntaje
Implementación correcta, sin errores de compilación y de acuerdo a restricciones	5
Comentarios aclaratorios	2,5
Provisión de datos de prueba	2,5

Ejercicio 2 (15p)

Levantar en VPL el código implementado en este ejercicio y las respuestas a las preguntas dejar como comentario en la actividad indicando a qué ítem corresponde la respuesta.

a) Dado el código de `RadixSort` en SL presentado aquí:
   a.1) Mencione que restricciones impone esta implementación (en SL).
   a.2) Indique la *T(n)* y la *O* de este algoritmo en el peor caso.

```
sub  RadixSort(ref V:vector[*] numerico )
tipos
     colaitem : registro {
                elems : vector[1500] numerico
                cont  : numerico
     }
var
  i,j,k,h,digito :numerico
  iter           :numerico
  continuar      :logico
  colas          : vector[10] colaitem
inicio
  iter = 1
  repetir
     colas = {{{0,...},1},...} /* Inicializar colas */
     continuar = NO
     desde i=1 hasta alen(V) {
        continuar = ( int(V[i]/iter)  > 10 ) or continuar
        digito    = int(V[i]/iter)  % 10 + 1
        colas[digito].elems[colas[digito].cont] = V[i]
        inc(colas[digito].cont)
     }
     iter = iter * 10
     j = 1
     desde i=1 hasta 10 {
        h = colas[i].cont -1
        desde k=1 hasta h {
            V[j] = colas[i].elems[k]
            inc(j)
        }
     }
  hasta ( not continuar )
fin
```

b) Implemente en Java un algoritmo *RadixSort* que pueda ordenar cadenas alfabéticas (base 26).

c) ¿En qué caso el algoritmo *RadixSort* puede tardar más que un tiempo lineal?

d) Implemente una versión de *RadixSort* similar al mostrado en la diapositiva de clase utilizando el algoritmo *countingSort*, también mostrado en esas láminas, como el algoritmo a ser utilizado por RadixSort para la ordenación en cada dígito. ¿Qué ventaja poseería sobre la versión de RadixSort presentado en el ítem a) ?

Rúbrica	**Puntaje**
Respuesta Correcta para (a.1)	2
Respuesta Correcta para (a.2)	1
Implementación correcta en (b)	3
Comentarios aclaratorios en (b)	2
Respuesta correcta en (c)	2
Implementación correcta en (d)	3
Comentarios aclaratorios en (d) y respuesta correcta a la pregunta.	2

Ejercicio 3 (10p) (Alzar en VPL, la fundamentación de tiempo colocar como comentario en el mismo código)

Considere un arreglo ***a*** de N elementos comparables $a_0, a_1, a_2, .., a_{N-1}$. Se dice que un arreglo ***b*** es circular de ***a*** si ***b*** consiste en el subarreglo $a_k, a_{k+1}, .., a_{N-1}$ seguido del subarreglo $a_0, a_1, a_2, ... , a_{k-1}$ para algún entero *k*. En el ejemplo de abajo, ***b*** es circular de ***a*** con *k=7* y *N=10*:

arreglo ordenado ***a[]***

0	1	2	3	4	5	6	7	8	9
1	2	3	5	6	8	9	34	55	89

arreglo circular ***b[]***

0	1	2	3	4	5	6	7	8	9
34	55	89	1	2	3	5	6	8	9

Suponga que usted tiene como entrada un arreglo circular ***b*** de algún arreglo ordenado (no tiene el valor de *k* ni el arreglo ordenado). Asuma que el arreglo ***b*** consiste en N elementos comparables, todos diferentes.

Diseñar una función **genérica** en **Java** que reciba un arreglo circular ***b*** de elementos comparables y un valor ***t*** de tipo genérico y retorne si ***t*** existe o no en el arreglo.

El algoritmo debe estar en *O(log N)* en el peor caso. *N* es el tamaño del arreglo. *Fundamentar el tiempo*.

Rúbrica	**Puntaje**
Implementación Correcta (sin errores de compilación)	2
Uso de generics	3
La implementación es O(log N) en el peor caso	3
Fundamentación de tiempo	2