# Actividad 1.4.1 Algoritmos de Ordenamiento

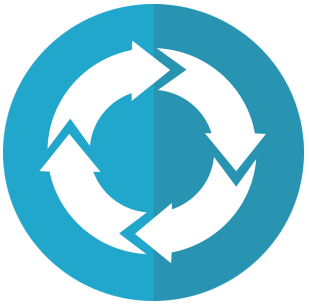
Tecnológico de Monterrey

Miguel Jiménez Padilla - A01423189
Silvio Emmanuel Prieto Caro - A01423341
Marco Antonio Gardida Cortés - A01423221
Alejandro Hidalgo Badillo - A01423412

**Profesora:**

Mónica Larre Bolaños

**9 de septiembre de 2021**



## Descripción del Problema

En esta ocasión se retoma el problema de los algoritmos de ordenamiento, esta vez utilizando los algoritmos de merge y quicksort. Se requiere ordenar un vector de objetos de tipo RFC, utilizando orden alfabético o numérico según sea el caso.
Un objeto de tipo RFC debe contener cuatro letras, seguidas de seis dígitos y por último otras tres letras. Estos se deben comenzar a acomodar tomando como referencia las primeras cuatro letras. Si todas las letras son iguales, se acomodan utilizando los dígitos y, por último, las letras del final.
El programa debe mostrar el vector ordenado utilizando ambos métodos, así como el tiempo de ejecución de cada uno.

## Casos de Prueba

```
EPKT804676EAO
FYWD804603YIN
GOPR570481QND
HDOL270875BFR
IGUI537385WQW
ILQS124171AMB
INHP220475TKU
IXEV514453IYD
IYFP355165UST
JDCI847460PXE
KEDG881125RRC
LPEP122151NJL
LQHC201820FME
MFRA375654YTM
MJWY286287YUE
NIWM227277CWX
NWCP146162RTJ
ODGU872105AIY
OIBA106438ONO
OXYL800818YBD
PDTM088182KKS
QACK842820XNE
QEXM347027FQQ
QQDT010556BII
QYUJ365568XAH
RRAL572814ECW
RWDK155415YQJ
TQIJ224214SOV
VBLM427825HXW
VBNI028171VTS
WCMH507066FSF
WFOG823556MIM
WHHK228672PMN
WILK372010BLC
WKVT278257VDI
WMEP603235VFG
XFLD067534UNR
XOLN710601VXX
XRVE328208DWJ
YACF223508HKW
YFTR403775GDP
YHLA722353IHU
------------------------------------------------------------
Algoritmo   Ejecución desordenado  Ejecución ordenado
MergeSort   0.00165                 0.001062
QuickSort   0.000657                 0.001364
QuickSort es más rápido cuando está desordenado
```

```
EDKP6215680BL
EEBP465863HLJ
EJXG647601UBY
EWHR367273XPP
FHIH846624RWJ
FRDQ886251QNM
GDFR428536FOE
GFWU271410FLF
GGGS407832MIG
GJAW805103UVN
GLIN5027870BS
HECV270252OLC
HQOJ081754YTC
HXGF515316QLO
IICX454532GKI
IIFI585038ADE
INDU127363CBA
ISWH622638RGK
JDYE751272KLU
KFBG387361JAO
KXNE100104AVI
LNDX715586TMC
LQIK041870WTC
MDQY442833QXD
MIBP421277QEV
MRUE786144YOU
NNHS210258CFO
OSUM351113XUG
PPES208625BCN
QGCU845285MIV
QIPE706261ITC
QOEN028014YYG
QSEK782660BAS
RNLA617565HFU
TECL700047TGX
UGVD227330VVI
ULMX568115BRO
VHYJ060164WAV
WFPA208463AAS
XXUB814760UKO
YDQE875214RME
YGBA442686LGR
------------------------------------------------------------
Algoritmo   Ejecución desordenado  Ejecución ordenado
MergeSort   0.001289                  0.001103
QuickSort   0.000513                  0.00162
QuickSort es más rápido cuando está desordenado
```

```
CJGX304623RVD
CLQR128478SWK
DHEA133110PVT
DNVX633002AMH
DUBI183282PHP
EECQ776372AAA
EMBB004341QMX
GJTL521582RNN
HDJH707237XMN
HHMF341486HLL
HWSC333216FGJ
HWWD278414JQW
JCMJ533077DPY
JCYB788660LJJ
JTIQ765640JHR
JWWE507181TLH
JWWQ632523HRQ
KORO547376XXF
LDVI406026CYV
MPNH646630XUG
MUGA087025BDE
NJGE475537NFO
NKCT331424DQY
NNCX572744XWR
OEMN847043XTY
OFAK781216WYE
ORCF673711XST
PGCG882333TVD
PJCJ388412DNW
QNOF538452WVF
QNUQ384331TIB
RENS402536YER
RGUQ354358WRG
RSYB872246FBO
SVYC754734HQA
UXNC830685VNM
VNDH3624830JU
VQND353501PXW
VUDP141771FLO
WCKP711225KAN
XRPL178787TUC
YAAF614431CJN
------------------------------------------------------------
Algoritmo   Ejecución desordenado  Ejecución ordenado
MergeSort   0.001072                  0.0012
QuickSort   0.00043                  0.001682
QuickSort es más rápido cuando está desordenado
```

**Análisis de Complejidad**

```
vector<RFC> merge(vector<RFC>& left, vector<RFC>& right)
{
    vector<RFC> result;
    while ((int)left.size() > 0 || (int)right.size() > 0) {
        if ((int)left.size() > 0 && (int)right.size() > 0) {
            if (left.front() < right.front()) {
                result.push_back(left.front());
                left.erase(left.begin());
            }
    else {
                result.push_back(right.front());//----------
                right.erase(right.begin());
            }
        }  else if ((int)left.size() > 0) {
                for (int i = 0; i < (int)left.size(); i++)
                    result.push_back(left[i]);
                break;
        }  else if ((int)right.size() > 0) {
                for (int i = 0; i < (int)right.size(); i++)
                    result.push_back(right[i]);
                break;
        }
    }
    return result;
}
```

Declaración de vector: 1
While:
- Comparacion + or: 2n
- Comparacion + and: 2n
- Comparación: 1n
- Acceso a funciones: 4n
- Comparación: 1n
- Primer For:
  - Declaración e igualación de variable: 2
  - Comparación: 1n
  - Incremento: 2n
  - Acceso a función: 1n
  - Break: 1n
  - **5n + 2**
- Comparación: 1n
- Segundo For:
  - Declaración e igualación de variable: 2
  - Comparación: 1n

Incremento: 2n
Acceso a función: 1n
Break: 1n
**5n + 2**

**(10n + 4)(11n) = 110n^2 + 44**

Return: 1

**110n^2 + 46**

Complejidad Cuadrática: O(n^2)

```
vector<RFC> mergeSort(vector<RFC>& m, double &timez)
{
  unsigned t0,t1;
  t0=clock();
   if (m.size() <= 1)
      return m;

   vector<RFC> left, right, result;
   int middle = ((int)m.size()+ 1) / 2;

   for (int i = 0; i < middle; i++) {
      left.push_back(m[i]);
   }

   for (int i = middle; i < (int)m.size(); i++) {
      right.push_back(m[i]);
   }

   left = mergeSort(left, timez);
   right = mergeSort(right, timez);
   result = merge(left, right);

   t1 =clock();
  timez = (double(t1-t0)/CLOCKS_PER_SEC);
   return result;
}
```

Declaración de variables: 2
Igualación de variable: 1
Comparación: 1
Return: 1
Declaración de vectores: 3
Declaración e igualación de variable: 3
Primer For:
    Declaración e igualación de variable: 2

Comparación: 1n
Incremento: 2n
Acceso a función: 1n
**4n + 2**

Segundo For:
Declaración e igualación de variable: 2
Comparación: 1n
Incremento: 2n
Acceso a función: 1n
**4n + 2**

Igualación de variables + acceso a funciones: 6
Igualación de Variable:  1
Declaracion más igualación: 2
Return: 1
**8n + 21**

Complejidad Lineal: O(n)

```
void quickSort(vector<RFC> &rfcs, int izq, int der, double &timex) {

 unsigned t0,t1;

 RFC piv = rfcs[izq];
 int i = izq;
 int j = der;
 RFC aux;

  //STARTS RUNNING
  t0=clock();

 while (i< j) {
  while ((rfcs[i]<=piv && i<j)) i++;
  while (rfcs[j] > piv) j--;
  if (i < j) {
   aux  = rfcs[i];
   rfcs[i] = rfcs[j];
   rfcs[j] = aux;
  }
 }

 rfcs[izq] = rfcs[j];
 rfcs[j] = piv;
 if (izq<j - 1)
  quickSort(rfcs, izq, j - 1,timex);
 if (j + 1 <der)
  quickSort(rfcs, j + 1, der,timex);

  //ENDS RUNNING
  t1=clock();
  timex = (double(t1-t0)/CLOCKS_PER_SEC);

}
```

Declaración de variables: 2
Declaración e igualación de variables: 6
Declaración de variable: 1
Igualación de variable: 1
While:
- Comparación: 1n
- Sub While 1:
  - Comparación: 1n
  - Comparaciones: 2n
  - Incremento: 2n
  - **5n**
- Sub While 2:
  - Comparación: 1n

Decremento: 2n

**3n**

Comparación: 1n

Igualación de variables: 3n

**40n^2**

Igualación de variables: 2

Comparacion: 1

Acceso a Función: 1

Comparacion: 1

Acceso a Función: 1

Igualación de Variable:  1

Declaracion más igualación: 2

**40n^2 + 19**

Complejidad Cuadrática: O(n^2)

```
void rellenarArray(vector<RFC> &rfcsQuick, vector<RFC> &rfcsMerge,int &length){
    length=MAX;
    string nombre,fecha,llave;

  //65-90 ASCII
  srand(time(nullptr));

  for (int i=0; i<MAX; i++){
    for(int z=0; z<13; z++){
      if (z<4){
        //RFC[z]=65 + rand() % (90-65);
        //RFC+= 65 + rand() % (90-65);
        nombre+= 65 + rand() % (90-65);
      }
      else if (z>=4 && z<10){
        //RFC[z]= rand() % 9;
        //RFC+=to_string(rand() % 9);
        fecha+= to_string(rand() % 9);
      }
      else{
        //RFC[z]=65 + rand() % (90-65);
        //RFC+= 65 + rand() % (90-65);
        llave+= 65 + rand() % (90-65);
      }

    }
    rfcsQuick.push_back(RFC(nombre,fecha,llave));
    rfcsMerge.push_back(RFC(nombre,fecha,llave));

    nombre="";
    fecha="";
    llave="";
  }
}
```

Igualación de Variable: 1
Declaración de variables: 3
Acceso a función: 1
For:
- Declaración e Igualación: 2
- Comparación: 1n
- Incremento: 2n
- SubFor:
  - Declaración e Igualación: 2
  - Comparación: 1n
  - Incremento: 2n
  - Comparación: 1n
  - Incremento más acceso a función: 2n
  - Comparaciones: 2n
  - Incremento más acceso a función: 2n

Incremento más acceso a función: 2n

**12n + 2**

**(12n + 2)(3n + 2) = 36n^2 + 30n + 4**

Acceso a funciones: 2

Igualación de variables: 3

**36n^2 + 30n + 14**

Complejidad Cuadrática: O(n^2)