FACULTAD DE INFORMÁTICA

UNIVERSIDAD DE MURCIA

DIVIDE Y VENCERÁS

Realizado por:

Daniel Ibáñez Molero 48658416F

Ginés Miguel Cánovas Méndez 23333196H

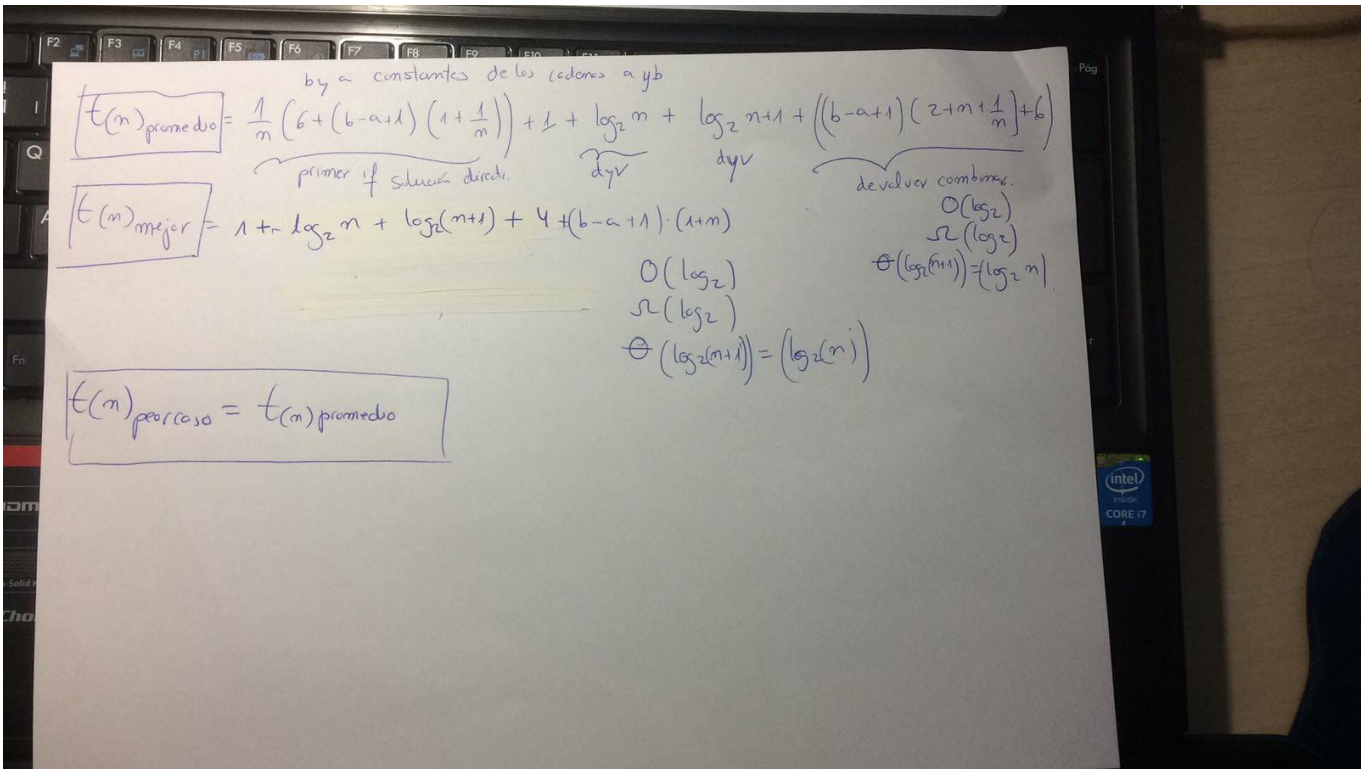
Pseudocódigo:

```
Programa ejDyV

tipo respuesta{
     i,j: entero
     numpares: entero
}
var
     C1,C2,C: cadena
     tamañorespuesta: entero
     tamañominimo: entero
     solucion: respuesta

//Pequeño detecta si la cadena tiene el tamaño mínimo
operacion Pequeño(i,j: entero) devuelve booleano
begin
     si (j-i <= tamañominimo)
          devuelve verdadero
     sino devuelve falso
     finsi
end

//solucionDirecta buscará la solución de forma lineal
en el caso minimo
operación solucionDirecta(i,j: entero) devuelve
respuesta
begin
     resultado: respuesta
     numpares, contador: entero
     contador := i
     numpares := -1
     repetir
          contador:=contador+1
          si (contenidoEnCadena(C1, cadena[contador])
y contenidoEnCadena(C2,cadena[contador+1]))
               numpares:= numpares+1
          finsi
     hasta (contador < j)
     resultado.i := i
     resultado.j := j
     resultado.numpares=numpares
     devolver resultado
end

//Función combinar devolverá la mejor cadena entre la
derecha, la izquierda y las que quedan entre estas.
operación combinar(cadena1,cadena2: respuesta)
devuelve respuesta
begin
     mejor: respuesta
     mejor.numpares := -1
     contador: entero := 0
     numpares: entero := 0
     caux: entero
     medioi, medioj: entero
```

```
     si (cadena1.numpares > mejor.numpares) mejor =
cadena1

     contador := cadena2.i-tamañorespuesta
     repetir
          contador:=contador+1
          repetir
               caux := contador
               si (contenidoEnCadena(C1,
cadena[caux]) y contenidoEnCadena(C2,cadena[caux+1]))
                    numpares:= numpares+1
               finsi
          hasta (caux < t-1)
          si (numpares > mejor.numpares)
mejor=[cadena desde contador hasta contador+t]
     hasta (contador = cadena2.i-1)

     si (cadena1.numpares > mejor.numpares) mejor =
cadena1

     devolver mejor
end

//Función divide y vencerás
divideyvenceras(i,j: entero) devuelve respuesta
begin
     si(pequeño(i,j)) devuelve solucionDirecta(i,j)

     m: entero
     m:= (i+j)/2
     respuesta solIzq = divideyvenceras(i,m)
     respuesta solDer = divideyvenceras(m+1,j)
     devuelve combinar(solIzq,solDer)
end

//Main del programa
main()
begin
     [Lee los datos por pantalla]
     i,j: entero
     i=0
     j=C.longitud()
     divideyvenceras(i,j)
     [Muestra los datos por pantalla con formato]
end
```

Este será el pseudocódigo en el que nos hemos basado para realizar nuestro programa. Incluye todas las funciones del esquema básico divide y vencerás: la principal, la solución directa y la función combinar. Nuestro ejercicio (el 18) nos pide que de una cadena C saquemos la cadena de tamaño m con más instancias de un carácter del conjunto c1 seguido de un carácter c2.
En nuestro caso directo recorremos la cadena de tamaño mínimo de forma lineal, comprobando el carácter y el siguiente a este. En la práctica hemos puesto de caso base 2, ya que es el tamaño mínimo en el que puede haber un par.
La función divide y vencerás aplica el esquema de recorrido, llamando a caso base si la cadena es la mínima o dividiendo l cadena y llamándose a si misma, llamando finalmente a la función combinar.
Por último la función combinar comparará que cadena es la de mayor valor, si la derecha, izquierda o alguna de las que se hallan en medio, y guardará la mejor.

Estudio teórico:



```
TIEMPO PROMEDIO Y MEJOR CASO:

T(n) = 1/n*(6 +(b −a + 1)*(1 + 1/n)) + 1 + log base2 (n) +
log base2 (n + 1) + ((b −a +1)*(2 + n + 1/n)+6)

O = log base2 (n)

Omega = log base2 (n)

o = log base2(n + 1)

TIEMPO MEJOR CASO:

T(n) = 1+ log base2 (n) + log base2 (n + 1) + 4+ (b −a +
1)* (1 + n)

O = log base2 (n)

Omega = log base2 (n)
```

Hemos realizado el análisis teórico de dicho algoritmo basándonos en las diferentes estructuras que están en él. Como podemos observar hemos sacado además diferentes ecuaciones en caso de que se produjese por un mejor tiempo y un tiempo algo grande. Al realizar dichas comprobaciones vemos que el tiempo en el peor caso es muy parecido por no decir casi igual al promedio ya que se cumplirían todas las condiciones que hay en nuestro algoritmo. Lo que nos sorprende es que el mejor tiempo tampoco suele ser muy diferente aunque desaparezcan algunas constantes en las condiciones de partida. Así mismo observamos que el tiempo de ejecución de nuestro algoritmo estará en torno a un log base2 (n).

Finalmente destacar que el algoritmo al llamarse la función diferentes veces nos produce una cosa llamada ecuaciones de recurrencia. Esas ecuaciones de recurrencia nos dará el valor de diferentes constantes para sacar siempre un valor diferente dependiendo de los parámetros de entrada.

A continuación hay realizado una parte hay realizado un estudio experimental que nos dará una idea de cómo funciona un log de estas características.

Programación del algoritmo:

```
#include <iostream>                //Realizado por Daniel
Ibáñez Molero y Ginés Miguel Cánovas Méndez
#include <string>
#include <math.h>                  //usado para rand().
#include <stdio.h>
#include <stdlib.h>
using namespace std;
```

```
struct respuesta{                    //Tipo auxiliar que
usaremos para representar nuestras cadenas respuesta.
     int i;                               //Comienzo de la
cadena respuesta.
     int j;                               //Fin de nuestra
cadena.
     int numpares;                  //Numero de pares en
nuestra cadena.
};
int n;                                    //Tamaño que tiene
nuestra cadena C
string C;                          //Cadena introducida en
la que buscar.
string C1;                                //Primer conjunto
del que sacar caracteres.
string C2;                                //Segundo conjunto
del que sacar caracteres.
const int m=2;                     //Tamaño mínimo.
int t;                                    //Tamaño que
tendrá nuestra cadena solución.

bool contenidoEnCadena(string cadena, char ch){
     //Funcion para hallar si el caracter ch está
contenido en el conjunto C.
     return cadena.find(ch) != string::npos;
}
bool pequenio(int i, int j){
     //FUNCION PEQUEÑO
     return j-i<=m;
     //Nuestro tamaño escogido es 2.
}
respuesta solucionDirecta(int i, int j){
     //FUNCION SOLUCIONDIRECTA
     respuesta directa = {0, 0, -1};
     //Halla de forma lineal la solucion en el caso
mínimo.
     int npares = 0;
     //EN nuestro caso, el caso mínimo será 2.
     for (int cont = i; cont < j;cont++){
          if (contenidoEnCadena(C1, C[cont]) && cont!=j){
               if(contenidoEnCadena(C2, C[cont+1])){
                    npares++;cont++;
               }
          }
     }
     directa.i = i;
     //Almacena la solución mínima.
     directa.j = j;
     directa.numpares = npares;
     return(directa);
     //Devuelve la solución mínima.
}
respuesta combinar(respuesta SolIzq,respuesta SolDer){
               //FUNCION COMBINAR
```

```
     respuesta mejor = {0, 0, -1};
                    //Compararemos que cadena tiene el
mayor numero de pares.
     if (SolIzq.numpares > mejor.numpares){
                    //Evaluamos la cadena izquierda.
          mejor.i = SolIzq.i;
          mejor.j = SolIzq.j;
          mejor.numpares = SolIzq.numpares;
     }

     int auxi = SolDer.i-t;int auxj = auxi+t-1;
                    //Los parametros Aux servirán para
delimitar
     for (int cont = 0; cont < t-1;cont++){
                    //las cadenas intermedias creadas en
bucle.
          int nparesaux = 0;
                         //Calculo y comparación de las
respuestas de en medio.
          auxi++;auxj++;
          if (auxi >= SolIzq.i && auxj <= SolDer.j){
               for (int caux = auxi; caux < auxj;caux++){
                    if (contenidoEnCadena(C1, C[caux]) &&
contenidoEnCadena(C2, C[caux+1])){
                         nparesaux++;
                         caux++;
                    }
               }
               if (nparesaux > mejor.numpares){
                    mejor.i = auxi;
                    mejor.j = auxj;
                    mejor.numpares = nparesaux;
               }
          }
     }

     if (SolDer.numpares > mejor.numpares){
                    //Evaluamos la cadena derecha.

          mejor.i = SolDer.i;
          mejor.j = SolDer.j;
          mejor.numpares = SolDer.numpares;
     }
     return(mejor);
                         //Devuelve el resultado.
}
respuesta dyv(int i,int j){
     //FUNCION DIVIDEYVENCERÁS
     if(pequenio(i,j)){
     //Analiza si el caso es Pequeño.
          return solucionDirecta(i,j);       //De serlo,
lo analiza directamente.
     }
     else{
     //Si no, divide la cadena
```

```
          int div = (i+j)/2;
          respuesta solIzq = dyv(i,div);
          respuesta solDer = dyv(div+1,j);
          return combinar(solIzq,solDer);         //Y
devuelve Combinar de ambas partes.
     }
}
int main(int argc, char const *argv[]) {
                    //CUERPO PRINCIPAL DEL PROGRAMA
     cout << "Introduzca el primer conjunto C1." << endl;
               //Lee y muestra los datos introducidos por
pantalla.
     cin >> C1;
     cout << "Introduzca el segundo conjunto C2." << endl;
     cin >> C2;
     cout << "Los conjuntos son "<< C1 << " y " << C2 <<
"." <<endl;
     cout << "Introduzca la longitud de la cadena C en la
que desea buscar los pares." << endl;
     cin >> n;
    srand(time(NULL));
    for(int i=0; i <n; i++){
          char caracter;
        caracter = 97 + rand() % (122 - 97);
          C = C + caracter;
     }
    cout << "Cadena: " << C << "." << endl;
     cout << "Introduzca el tamaño de la respuesta
deseado." << endl;
    cin >> t;
     if (t > C.length()){
          cout << "[!]ERROR: t no puede ser mayor que el
tamaño de la cadena." << endl;
          cout << ">En lugar de " << t << " tomaremos para
t el mayor tamaño posible: " << C.length() << "." << endl;
          t = C.length();
     }

     respuesta cadena = dyv(0, C.length()-1);
                    //Aplica divide y vencerás.
     cout << "Subcadena: ";
                         //Muestra en pantalla el
resultado obtenido.
     for (int cont = cadena.i; cont < cadena.j+1;cont++)
cout << C[cont];
     cout << "." << endl;
     cout << "Contiene: " << cadena.numpares << " pares de
caracteres." << endl;
     cout << "La subcadena hallada comienza en la
posicion: " << cadena.i+1 << "º." << endl;
return 0;
}
```

Validación del algoritmo:

Para la realización del algoritmo hacemos uso de std::cout para que el propio programa nos muestre como está recorriendo la estructura de datos, así como la evolución de los datos tratados y la solución.

Una vez tenemos esto comprobamos varios casos independientemente, asignando un valor no aleatorio a C compuesto por “a”, “b” y “x”, con C1 = “a” y C2 = “b”, a fin de comprobar que todas las posibilidades sean correctas. También hemos comprobado que reconoce los caracteres de los conjuntos correctamente.
Unas cuantas cadenas en las que hemos comprobado el funcionamiento son:
ababxxxx con t respuesta=4 (Solución al principio)
xxxxabab con t respuesta = 4(Solución al final)
xxababxx con t respuesta = 4(Solución en medio, dividida en partes iguales)
xababxxx con t respuesta = 4 (Solución en medio, descompensada)
ababxxxx con t respuesta = 3(Solución con nº impar de caracteres, para comprobar que el conteo de pares se realiza solo en la subcadena)

Estudio experimental:

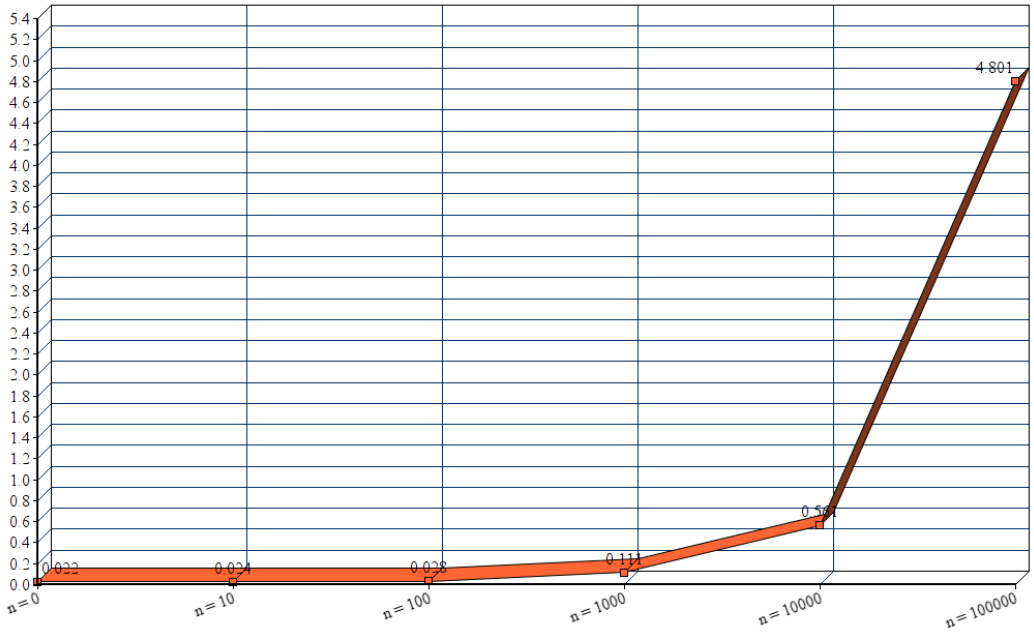
Para realizar el estudio del tiempo de ejecución usaremos el siguiente código:

```
#include<ctime>
int start_s=clock();

//Ejecución de nuestro algoritmo

int stop_s=clock();
cout<<(stop_s-start_s)/double(CLOCKS_PER_SEC)*1000;
```

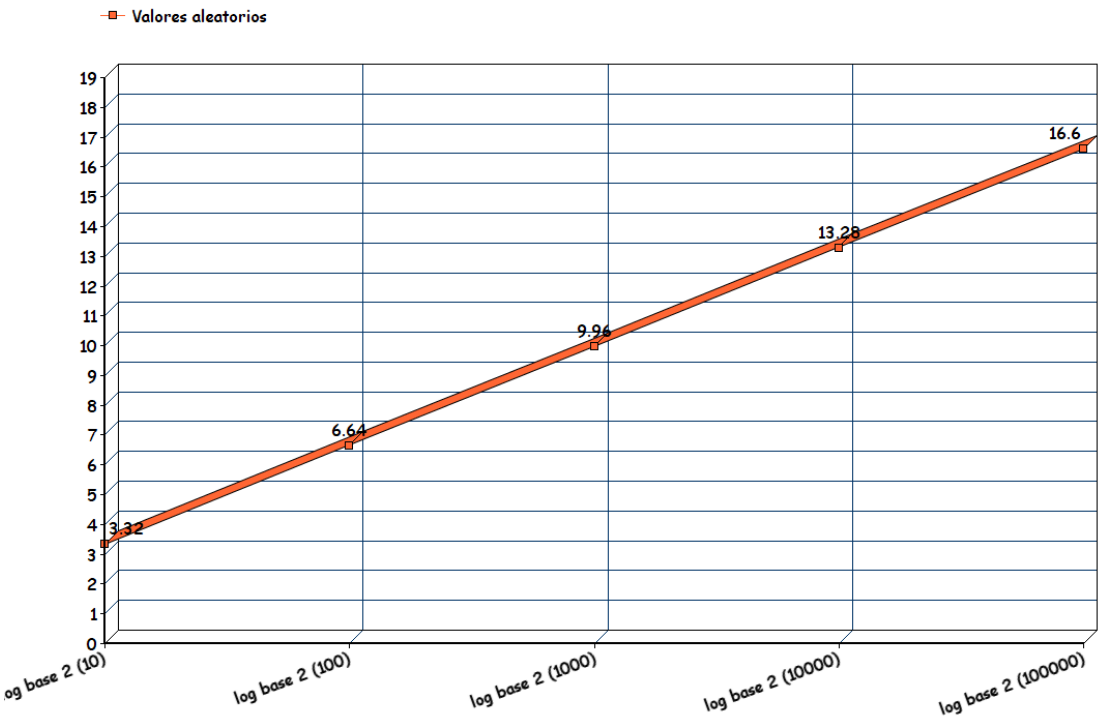
Lo cual nos devuelve los milisegundos que tarda su ejecución. Para simplificar, los conjuntos tomarán siempre el valor “abc” para C1 y “def” para C2 en el análisis. Además, el valor de nuestra solución pedido será siempre igual a 2.

Experimentos realizados:

Valor de n	Tiempo de ejecución
0	0.008
10	0.024
100	0.028
1000	0.111
10000	0.561
100000	4.801



Conclusiones:



Nuestro orden teórico no coincide con el orden práctico. Creemos que esto se debe en gran medida a que en nuestro pseudocódigo el hallar si un elemento se encuentra en un conjunto lo hemos planteado con una sola orden, cuando en verdad esa orden tendrá que recorrer este, con lo que el tiempo real aumenta de forma exponencial.

En relación al resto de trabajos, divide y vencerás (al menos en nuestra tarea asignada) tiene un tiempo de ejecución aceptable, aunque estamos bastante seguros que nuestro problema sería tratado mas eficientemente con un recorrido lineal.