Algoritmo Greedy

Adquisición Diferida
Problema de los Recipientes

PROBLEMA DE LA ADQUISICIÓN DIFERIDA

Adquisición diferida - Elementos

Conjunto de candidatos (C): conjunto de todos los equipos (cada equipo está representado por su incremento).

Conjunto de seleccionados (S): Equipos seleccionados.

Función solución (seHanCompradoTodosLosEquipos): la solución se obtiene una vez se han comprado todos los equipos.

Función de factibilidad: todos los candidatos seleccionados son factibles, por lo que no hace falta función de factibilidad.

Función selección (seleccionarEquipoMaxIncremento): el candidato a seleccionar en cada momento es el que tenga mayor incremento.

Función solución: devuelve el dinero total gastado al transcurrir los *t* meses.

Adquisición diferida - Pseudocódigo

```
S=Ø;
precio_total = 0;
num_meses_pasados = 0;

mientras ! seHanCompradoTodosLosEquipos()
    x = seleccionarEquipoMaxIncremento(C);
    C.eliminar(x);
    precio = calcularPrecio (x, num_meses_pasados)
    precio_total += precio;
    S.añadir(x);
    num_meses_pasados++;
fin
Devolver total;
```

Adquisición diferida - Optimalidad

Sea $(x_1, x_2,...,x_n)$ los equipos seleccionados por nuestro algoritmo con un precio total.

$$\text{Coste} = \sum_{t=0;\, i=0}^{n} 1000(r_i)^t = 1000 \sum_{t=0;\, i=0}^{n} (r_i)^t = 1000 \sum_{t=0}^{n} (r_t)^t$$

Supongamos que hay otra ordenación intercambiando dos elementos x_i y x_j cuyo coste total es menor:

$$1000\Big(\sum_{t=0}^{i-1} r_t^t + r_i^i + r_j^j + \sum_{t=j+1}^{n} r_t^t\Big) > 1000\Big(\sum_{t=0}^{i-1} r_t^t + r_j^i + r_i^j + \sum_{t=j+1}^{n} r_t^t\Big)$$

$$\Big(\sum_{t=0}^{i-1} r_t^t + r_j^i + r_i^j + \sum_{t=j+1}^{n} r_t^t\Big) > \Big(\sum_{t=0}^{i-1} r_t^t + r_j^i + r_i^j + \sum_{t=j+1}^{n} r_t^t\Big)$$

$$r_i^i + r_j^j > r_j^i + r_i^j$$

De los que obtenemos $r_i > r_j$ y $i < j$, pero por la naturaleza de nuestro algoritmo, éste hubiera seleccionado r_i antes que r_j => **contradicción**.

EL PROBLEMA DE LOS RECIPIENTES

Recipientes- Elementos

Conjunto de candidatos (C): Todos los objetos a meter en los recipientes.

Conjunto de seleccionados (S): Objetos en el orden en el que se van seleccionando.

Función solución (seHanSeleccionadoTodosLosObjetos): El algoritmo finaliza tras seleccionar todos los objetos.

Función de factibilidad: todos los candidatos seleccionados son factibles, por lo que no hace falta función de factibilidad, ya que el número de recipientes es ilimitado.

Función selección: definimos dos funciones de selección:

1. SeleccionaMayorPesoQueCabe(): Selecciona el objeto de mayor peso que cabe en el recipiente actual, devolviendo su posición, si ninguno cabe devuelve -1.

2. SeleccionarSiguienteObjetoQueCabe(): Selecciona el primer objeto del conjunto de candidatos que cabe en el recipiente actual devolviendo su posición, es decir, comprueba si el primer elemento cabe, si no pasa al siguiente y así sucesivamente. Si ninguno cabe devuelve -1.

Función solución: devuelve el número total de recipientes usados al seleccionar todos los objetos.

Recipientes - Pseudocódigo

```
S=Ø;
recipientes_usados = 1;
restante_actual = 1;

mientras quedanObjetos()
    pos = FuncionSeleccion(C);
    si (pos != -1)
        restante_actual -= C[pos];
        S.añadir(pos);
        C.eliminar(pos);
    sino
        restante_actual = 1;
        recipientes_usados++;
    fin
fin

Devolver recipientes_usados;
```

Recipientes- Función Greedy

```
int funcionGreedy(){

	int recipientes = 1, posObjeto;
	double restante_actual = 1;

	while( !C.empty() ){ //Mientras quedan candidatos (Funcion solución)

	  posObjeto = seleccionarElemento(restante_actual); //Función selección

	  if(posObjeto != -1){
		restante_actual -= C[posObjeto];
		S.push_back(C[posObjeto]);
		C.erase(C.begin() + posObjeto);
	  }else{
		recipientes++;
		restante_actual = 1;
	  }

	}

	return recipientes;
}
```

Funciones

SeleccionMayorQuecabe()

```
int seleccionarMayorPesoQueCabe(double pesoRestante){
	//En el caso de que no haya un objeto que quepa
	int pos = -1;
	double maximoPeso = 0;
	for(int i = 0; i < C.size(); i++)
	if(C[i] <= pesoRestante && C[i] >= maximoPeso){
		pos = i;
		maximoPeso = C[i];
	}
	//Si hay un objeto que cabe y tiene mayor peso
del que se habia guardado, se reemplaza.
	return pos;
}
```

selección

SeleccionarSiguienteObjetoQueCabe()

```
int seleccionarSiguiente(double pesoRestante){
	bool cabe = false;
	int pos = -1;

	for (int i = 0; i < C.size() && !cabe; i++)
	if (C[i] <= pesoRestante ){
		pos = i;
		cabe = true;
	}
	return pos;
}
```

Comparación de ejecución Algoritmos Greedy

Ejecución-Fuerza Bruta

```
$ ./recipientes-fuerzabruta 11
  Los pesos son:
  0.866929  0.471104  0.847393  0.134526  0.978849  0.51658
  0.161067  0.0484861  0.905923  0.847063  0.59071
  tiempo: 1.75205
  Se usan 7 recipientes
  La distribucion es:
   0.866929  en recipiente 1
   0.471104  en recipiente 2
   0.847393  en recipiente 3
   0.134526  en recipiente 3
   0.978849  en recipiente 4
   0.51658   en recipiente 2
   0.161067  en recipiente 5
   0.0484861 en recipiente 1
   0.905923  en recipiente 6
   0.847063  en recipiente 7
   0.59071   en recipiente 5
```

Funciones selección

SeleccionMayorQuecabe()

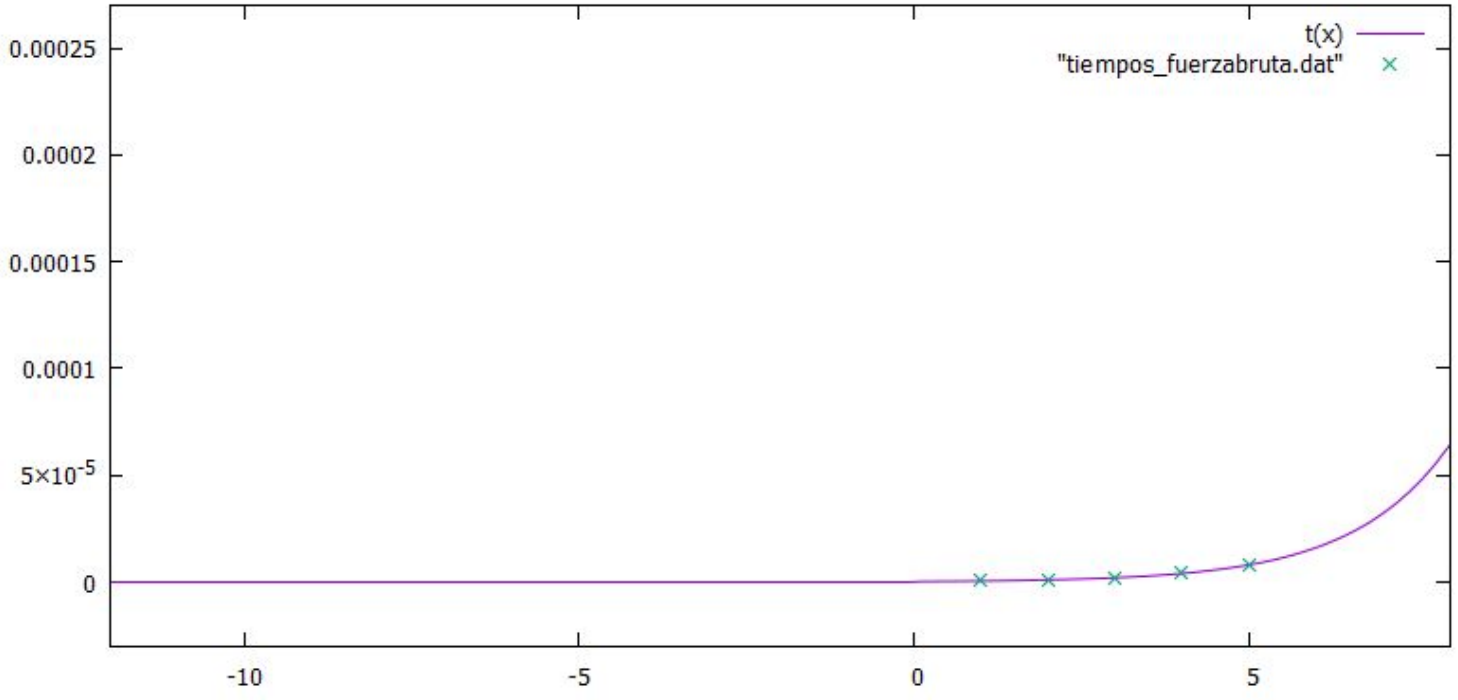
```
$ ./recipientes-greedy1 11
  Vector inicial:
  0.866929 0.471104 0.847393 0.134526 0.978849 0.51658
  0.161067 0.0484861 0.905923 0.847063 0.59071
  10 1.5e-05
  Recipientes: 7
  La distribución es:
   0.978849  en recipiente 1
   0.905923  en recipiente 2
   0.0484861 en recipiente 2
   0.866929  en recipiente 3
   0.847393  en recipiente 4
   0.134526  en recipiente 4
   0.847063  en recipiente 5
   0.59071   en recipiente 6
   0.161067  en recipiente 6
   0.51658   en recipiente 7
   0.471104  en recipiente 7
```

SeleccionarSiguienteObjetoQueCabe()

```
$ ./recipientes-greedy2 11
  Vector inicial:
  0.866929 0.471104 0.847393 0.134526 0.978849 0.51658
  0.161067 0.0484861 0.905923 0.847063 0.59071
  10 1.5e-05
  Recipientes: 8
  La distribución es:
   0.866929  en recipiente 1
   0.0484861 en recipiente 1
   0.471104  en recipiente 2
   0.134526  en recipiente 2
   0.161067  en recipiente 2
   0.847393  en recipiente 3
   0.978849  en recipiente 4
   0.51658   en recipiente 5
   0.905923  en recipiente 6
   0.847063  en recipiente 7
   0.59071   en recipiente 8
```

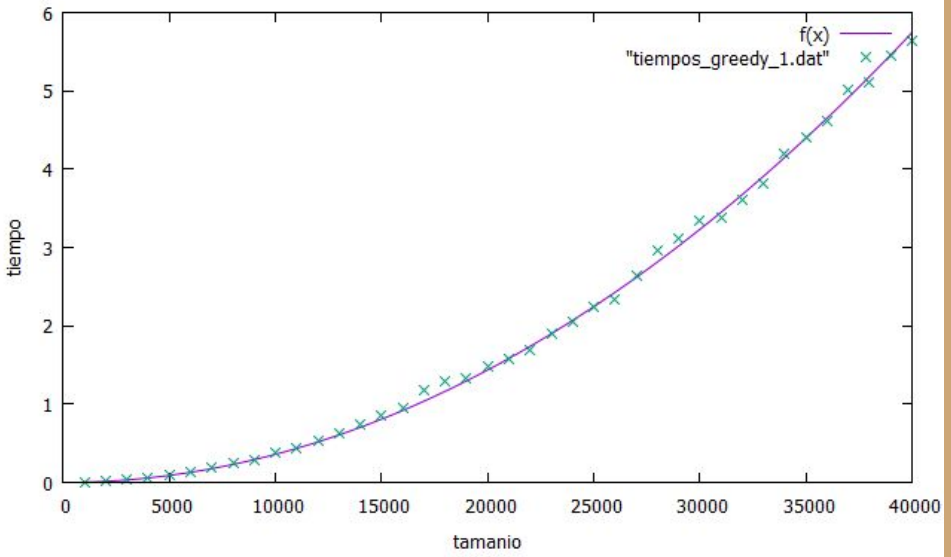
RESULTADOS OBTENIDOS
Algoritmos Greedy

Eficiencia híbrida del algoritmo fuerza bruta



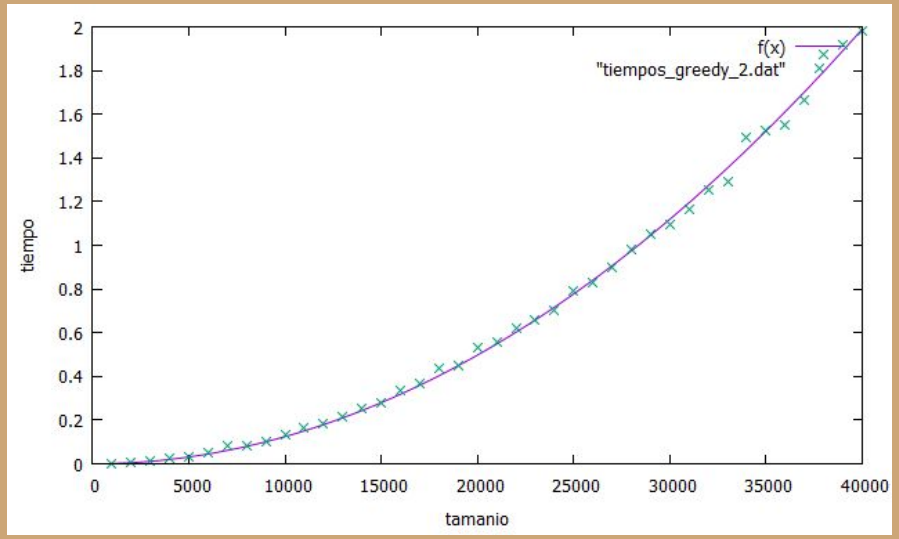
EFICIENCIA

Algoritmo SeleccionaMayorPesoQueCabe

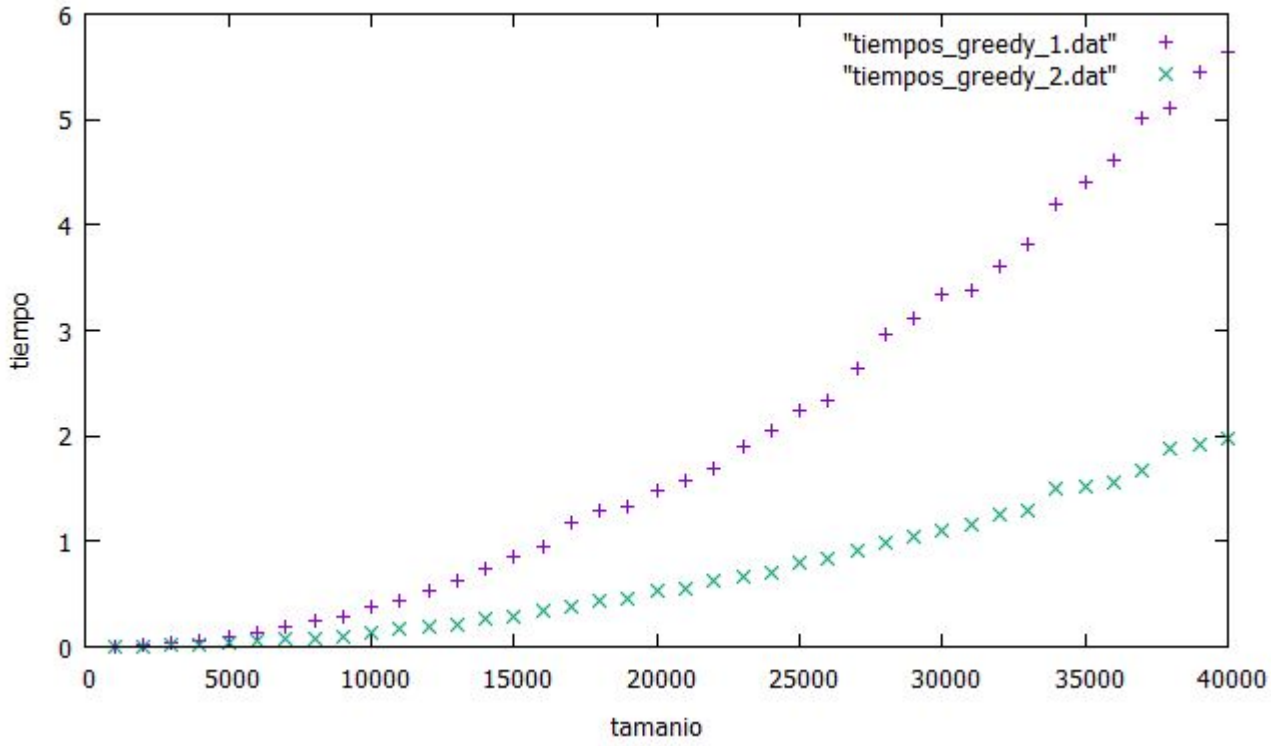


HÍBRIDA

Algoritmo SeleccionarSiguienteObjetoQueCabe



Comparación ef. empírica de los algoritmos Greedy



Comparación eficiencia empírica de los 3 algoritmos