Algoritmos Básicos

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2019-II

Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

CONTENTS

Búsqueda lineal

```
1 int linear_search(int X, const vector<int>& v){
2   for(int i = 0; i < v.size(); ++i){
3     if(v[i] == X)
4       return i;
5   }
6   return -1;
7 }
```

Búsqueda lineal

1. Implemente la función en un programa, genere un vector aleatorio y pruébela
2. Estime el tiempo de búsqueda promedio del 1 usando 100 vectores distintos aleatorios de 1×10^6 elementos c/u
3. Encuentre el tiempo de búsqueda del 1 en un vector de 1×10^6 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de búsqueda del 999999 en un vector de 1×10^6 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)

Búsqueda binaria

```
1  int binary_search(int X, const vector<int>& v){
2    int left = 0, right = v.size()-1;
3    while(left <= right){
4      int mid = (right+left) / 2;
5      if(v[mid] == X)
6        return mid;
7      else{
8        if(X > v[mid])
9          left = mid+1;
10       else
11         right = mid-1;
12     }
13   }
14   return -1;
15 }
```

Búsqueda binaria

1. Implemente la función en un programa y pruébela
2. Encuentre el tiempo de búsqueda del 1 en un vector de 1×10^6 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de búsqueda del 999999 en un vector de 1×10^6 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)

CONTENTS

Bogosort

```
1  bool is_sorted(const vector<int>& v){
2    for(int i = 0; i < v.size()-1; ++i){
3      if(v[i] > v[i+1])
4        return false;
5    }
6    return true;
7  }
8
9  void bogosort(vector<int>& v, minstd_rand0& rng){
10   while(!is_sorted(v)){
11     shuffle(v, rng);
12   }
13 }
```

Bogosort

1. Implemente la función en un programa y pruébela con vectores de 10, 12 y 14 elementos
2. Encuentre el tiempo de ordenamiento de un vector de 12 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de ordenamiento de un vector de 12 elementos aleatorio (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de ordenamiento de un vector de 12 elementos ordenado de mayor a menor (repita 100 veces el cálculo para calcular un promedio)

Ordenamiento por selección

```
1  void selection_sort(vector<int>& v){
2    for(int i = 0; i < v.size()-1; ++i){
3      int min_id = i;
4      for(int j = i+1; j < v.size(); ++j){
5        if(v[j] < v[min_id])
6          min_id = j;
7      }
8      swap(i, min_id, v);
9    }
10 }
```

Ordenamiento por selección

1. Implemente la función en un programa y pruébela con vectores cortos
2. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos aleatorio (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de mayor a menor (repita 100 veces el cálculo para calcular un promedio)

Ordenamiento de burbuja

```
1  void  bubble_sort(vector<int>& v){
2    bool swap_used = true;
3    while(swap_used){
4      swap_used = false;
5      for(int i = 0; i < v.size()-1; ++i){
6        if(v[i] > v[i+1]){
7          swap(i,i+1,v);
8          swap_used = true;
9        }
10     }
11   }
12 }
```

Ordenamiento de burbuja

1. Implemente la función en un programa y pruébela
2. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos aleatorio (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de mayor a menor (repita 100 veces el cálculo para calcular un promedio)

Ordenamiento por inserción

```
1  void insertion_sort(vector<int>& v){
2    for(int i = 1; i < v.size(); ++i){
3      int a = i;
4      int b = i-1;
5      while(v[b] > v[a]){
6        swap(a,b,v);
7        a--;
8        b = a-1;
9      }
10   }
11 }
```

Ordenamiento por inserción

1. Implemente la función en un programa y pruébela
2. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de menor a mayor (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos aleatorio (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de ordenamiento de un vector de 1000 elementos ordenado de mayor a menor (repita 100 veces el cálculo para calcular un promedio)

CONTENTS

Selección lenta

Usamos ordenamiento por selección:

```
1  int slow_selection(int k, vector<int>& v){
2    for(int i = 0; i < k-1; ++i){
3      int min_id = i;
4      for(int j = i+1; j < v.size(); ++j){
5        if(v[j] < v[min_id])
6          min_id = j;
7      }
8      swap(i, min_id, v);
9    }
10   return v[k-1];
11 }
```

Selección lenta

1. Implemente la función en un programa y pruébela
2. Encuentre el tiempo de selección del menor elemento en un vector de 1000 elementos sin repeticiones (repita 100 veces el cálculo para calcular un promedio)
3. Encuentre el tiempo de selección del mayor elemento en un vector de 1000 elementos sin repeticiones (repita 100 veces el cálculo para calcular un promedio)
4. Encuentre el tiempo de selección del elemento 500 en un vector de 1000 elementos sin repeticiones (repita 100 veces el cálculo para calcular un promedio)