UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

DIPARTIMENTO DI INFORMATICA

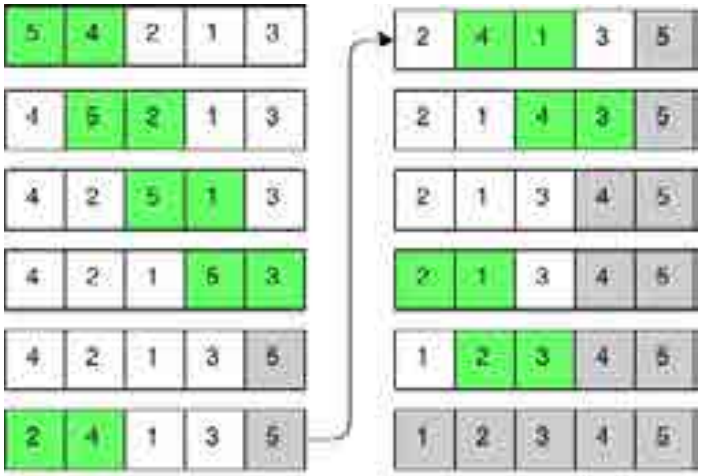
KDDE

Ordinamento e ricerca

Dott. Emanuele Pio Barracchia

Bubble sort

- In informatica il **Bubble sort**, o ordinamento a bolle, è un algoritmo di ordinamento che confronta ogni coppia di elementi adiacenti e li inverte nel caso in cui siano nell'ordine sbagliato.



- Il nome dell'algoritmo deriva dal modo in cui gli elementi vengono ordinati in questa lista: quelli più **"leggeri" "risalgono"** verso un'estremità, mentre quelli più **"pesanti" "affondano"** verso l'estremità opposta.

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 0
j = 0

21	10	3	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 0

21	10	3	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 0

21	10	3	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 1

10	21	3	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 1

10	21	3	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 2

10	3	21	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 2

10	3	21	8	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1

j = 3

10	3	8	21	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 1
j = 3

10	3	8	21	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2
j = 0

10	3	8	21	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2
j = 0

10	3	8	21	30

KDDE

Bubble sort - Algoritmo

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

i = 2

j = 1

3	10	8	21	30

Bubble sort - Algoritmo

Array iniziale:

21	10	3	8	30

Array dopo l'esecuzione dell'algoritmo **Bubble sort**:

3	8	10	21	30

KDDE

Bubble sort - Algoritmo

Senza puntatore

```
void bubblesort(int array[], int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                sorted = 0;
            }
        }
    }
}
```

Con puntatore

```
void bubblesort_pointer (int* pointer, int size){
    int i = 0;
    int j = 0;
    int sorted = 0; //sorted = false

    while (i < size && sorted == 0){
        sorted = 1;
        i++;
        for (j = 0; j < size-i; j++){
            if (*(pointer+j) > *(pointer+j+1)){
                int temp = *(pointer+j);
                *(pointer+j) = *(pointer+j+1);
                *(pointer+j+1) = temp;
                sorted = 0;
            }
        }
    }
}
```

Ricerca all'interno di un array

Esistono due algoritmi principali di ricerca di un elemento all'interno di un array:

- Algoritmo di ricerca sequenziale
- Algoritmo di ricerca dicotomica

Ricerca sequenziale

- L'algoritmo di ricerca sequenziale (o lineare) è un algoritmo utilizzabile per trovare un elemento in un insieme non ordinato
- Effettua una ricerca all'interno dell'array in modo sequenziale
- In particolare, controlla in sequenza gli elementi dell'array e verifica per ogni elemento controllato se è uguale all'elemento cercato
- L'algoritmo termina se:
  - L'elemento analizzato è l'elemento cercato
  - Ha controllato tutto l'array senza trovare l'elemento cercato

KDDE

Ricerca sequenziale

```
void linear_search(int array[],int search){
    int i = 0;
    int dim = sizeof(array)/sizeof(array[0]);

    for (i = 0; i < dim; i++)
    {
        if (array[i] == search)
        {
            printf("%d trovato in posizione %d.\n", search, i+1);
            break;
        }
    }
    if (i == dim)
        printf("%d non è presente nell'array.\n", search);
}
```

KDDE

Ricerca sequenziale - Versione ricorsiva

```c
#include<stdio.h>

int recSearch(int arr[], int left, int right, int x)
{
    if (right < left)
        return -1;
    if (arr[left] == x)
        return left;
    return recSearch(arr, left+1, right, x);
}

int main()
{
    int  search, n;

    printf("Inserire il numero di elementi dell'array: ");
    scanf("%d",&n);
    int array[n];

    printf("Inserire %d numeri: \n", n);

    for (int i = 0; i < n; i++)
        scanf("%d", &array[i]);

    printf("Inserire il numero da cercare: ");
    scanf("%d", &search);


    int index = recSearch(array, 0, n-1, search);

    if (index != -1)
        printf("L'elemento %d è presente in posizione %d. \n", search, index);
    else
        printf("L'elemento %d non è presente.", search);
    return 0;
}
```

Ricerca dicotomica

- L'algoritmo di ricerca dicotomica (o ricerca binaria) è un algoritmo di ricerca che individua l'indice di un determinato valore presente all'interno di un array **ordinato**

- Effettua mediamente meno confronti rispetto ad una ricerca sequenziale

- Il metodo alla base della ricerca binaria è lo stesso che si utilizza quando si cerca una parola all'interno del dizionario. L'idea è quella di iniziare la ricerca non dal primo elemento, ma da quello centrale:
  - Se l'elemento corrisponde a quello cercato, la ricerca termina
  - Se è superiore, la ricerca viene ripetuta sugli elementi precedenti
  - Se è inferiore, la ricerca viene ripetuta su quelli successivi

KDDE

Ricerca dicotomica

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 2

Last = 4

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 2

Last = 4

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 2

Last = 4

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 2

Last = 4

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 2

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 0

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 0

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 0

Middle = 0

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 0

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 1

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 1

Last = 1

Search = 5

1	5	8	10	12

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 1

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 1

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

Dott. E.P. Barracchia

KDDE

Ricerca dicotomica

First = 1

Middle = 1

Last = 1

Search = 5

```
void binary_search(int array[], int search){
    int dim = sizeof(array)/sizeof(array[0]);
    int first = 0;
    int last = dim - 1;
    int middle = (first+last)/2;

    while (first <= last) {
        if (array[middle] < search)
            first = middle + 1;
        else if (array[middle] == search) {
            printf("%d trovato in posizione %d.\n", search, middle+1);
            break;
        }
        else
            last = middle - 1;

        middle = (first + last)/2;
    }
    if (first > last)
        printf("%d non è presente nell'array.\n", search);

}
```

1	5	8	10	12

5 trovato in posizione 2.

KDDE

Ricerca dicotomica - Versione ricorsiva

```
void binary_search_rec(int array[], int low, int high, int search)
{
    int mid;

    if (low > high)
    {
        printf("%d non trovato.\n", search);
        return;
    }
    mid = (low + high) / 2;
    if (array[mid] == search)
    {
        printf("%d trovato in posizione %d.\n", search, mid);
    }
    else if (array[mid] > search)
    {
        binary_search_rec(array, low, mid - 1, search);
    }
    else if (array[mid] < search)
    {
        binary_search_rec(array, mid + 1, high, search);
    }
}
```