

Ricerca Binaria

Confronto tempo di esecuzione fra
metodo iterativo e ricorsivo

Condizioni iniziali

- Per evitare che il tempo di esecuzione dipenda dall'input inseriamo all'interno del codice il numero n di elementi che contiene il vettore e gli assegniamo dei valori tramite un ciclo for che genera n interi consecutivi
- Il valore da ricercare è inserito all'interno del codice
- Per calcolare il tempo di esecuzione utilizziamo la funzione `clock()` appartenente alla libreria `time.h`

Ricerca binaria iterativa (main)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int ric_b(int *,int,int);

int main(void)
{
    int *v,n,i,num;
    clock_t t;

    t = clock(); //inizializzazione clock se viene posizionato
    prima della chiamata della funzione ric_b il clock
    risulterà sempre 0
    n=1000000;

    v=malloc(n*sizeof(int)); //allocazione memoria vettore
    for(i=0; i<n; i++) //assegnazione valore
    {
        v[i]=i;
    }

    //se viene tolta la stampa il clock risulterà sempre 0
    //tranne per un range circa 1.000.000:100.000.000
    /*for(i=0; i<n; i++)
```

```
{
    printf("%d\n",v[i]);
}*/

printf("\n%d elementi\n",n);
num=145;
printf("\nelemento da ricercare: %d\n",num);

// t = clock();

if(ric_b(v,num,n)!=-1) printf("l'elemento si trova al
%d^ posto\n",ric_b(v,num,n));
else printf("elemento non trovato\n");

t = clock()-t;

printf ("\nci ho messo %d clicks (%f
secondi)\n",t,((float)t)/CLOCKS_PER_SEC);
free(v); //deallocazione memoria

system("PAUSE");
return 0;
}
```

Funzione iterativa

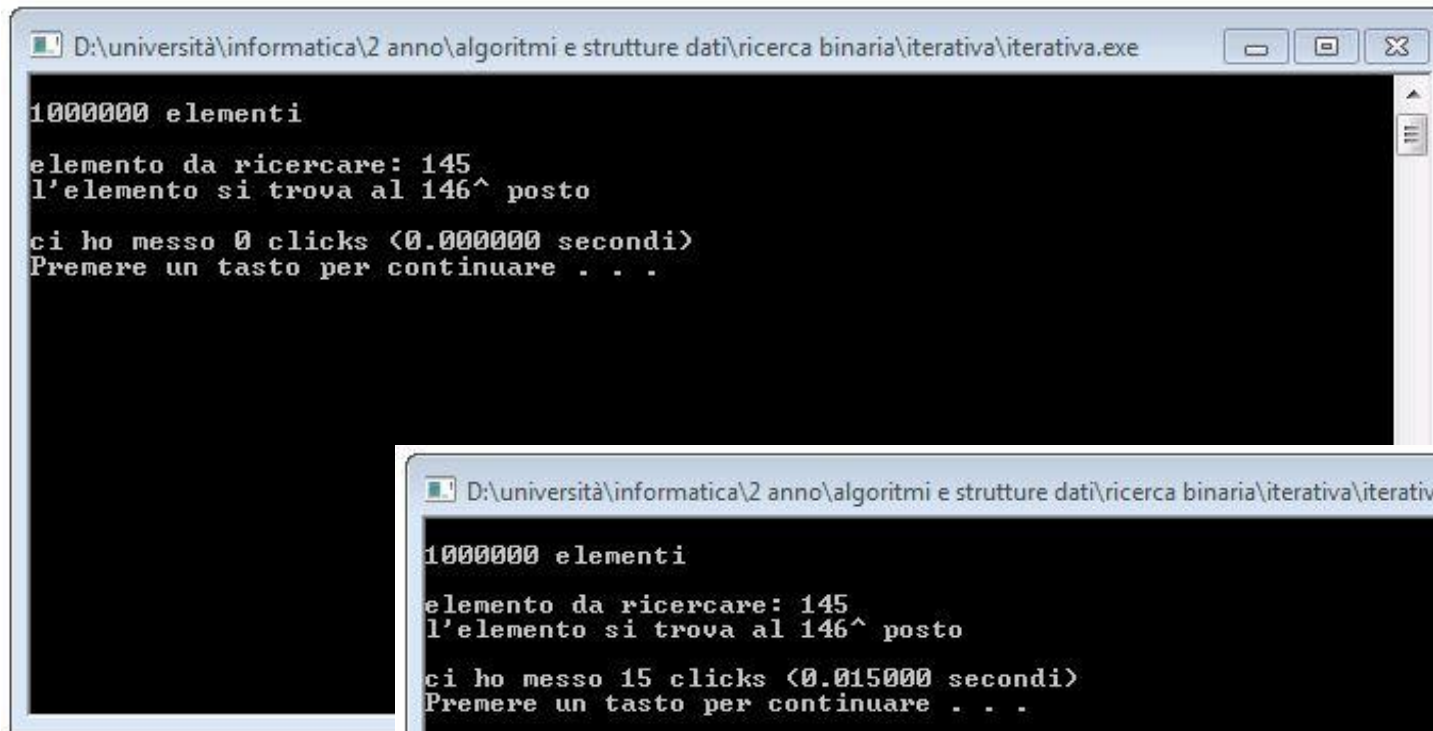
```
int ric_b(int *vett,int ele, int dim)
{
    int i=0,f=dim-1,m;
    while(i<=f)
    {
        m=(i+f)/2;
        if(vett[m]==ele) return m+1;
        if(vett[m]<ele) i=m+1;
        else f=m-1;
    }
    return -1;
}
```

Tempi di esecuzione per un range: 1.000.000-100.000.000

- Se inseriamo un numero di elementi inferiore a 1.000.000 il clock restituirà sempre un valore pari a 0.
- Facciamo 3 serie di prove ognuna che ricerca 4 valori diversi come si vede dalla tabella riassuntiva:
- I risultati non sono sempre gli stessi
- Non stampiamo gli elementi del vettore (vedremo che stampandoli i tempi di esecuzione aumenteranno considerevolmente)

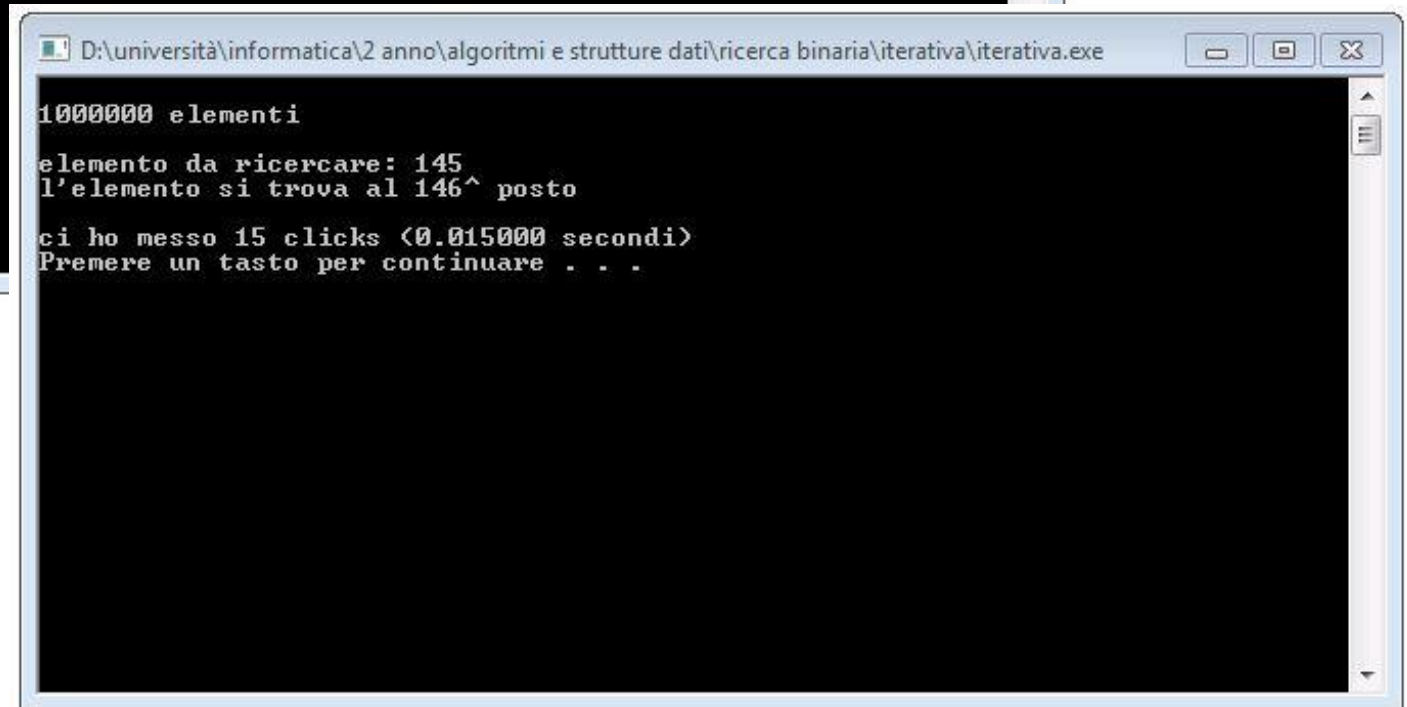
| Ricerca Binaria iterativa | | | |
|---------------------------|-------------|-------------|---------------------|
| | n° elementi | 1.000.000 | numero da ricercare |
| | clicks | secondi | |
| 1 | 0 | 0 | 145 |
| 2 | 15 | 0,015 | 14.852 |
| 3 | 15 | 0,015 | 1.245.785 |
| 4 | 15 | 0,015 | 105.123.458 |
| | n° elementi | 10.000.000 | |
| 1 | 31 | 0,031 | 145 |
| 2 | 31 | 0,031 | 14.852 |
| 3 | 31 | 0,031 | 1.245.785 |
| 4 | 31 | 0,031 | 105.123.458 |
| | n° elementi | 100.000.000 | |
| 1 | 280 | 0,28 | 145 |
| 2 | 280 | 0,28 | 14.852 |
| 3 | 296 | 0,296 | 1.245.785 |
| 4 | 312 | 0,312 | 105.123.458 |

I risultati non sono sempre gli stessi:



```
D:\università\informatica\2 anno\algoritmi e strutture dati\ricerca binaria\iterativa\iterativa.exe

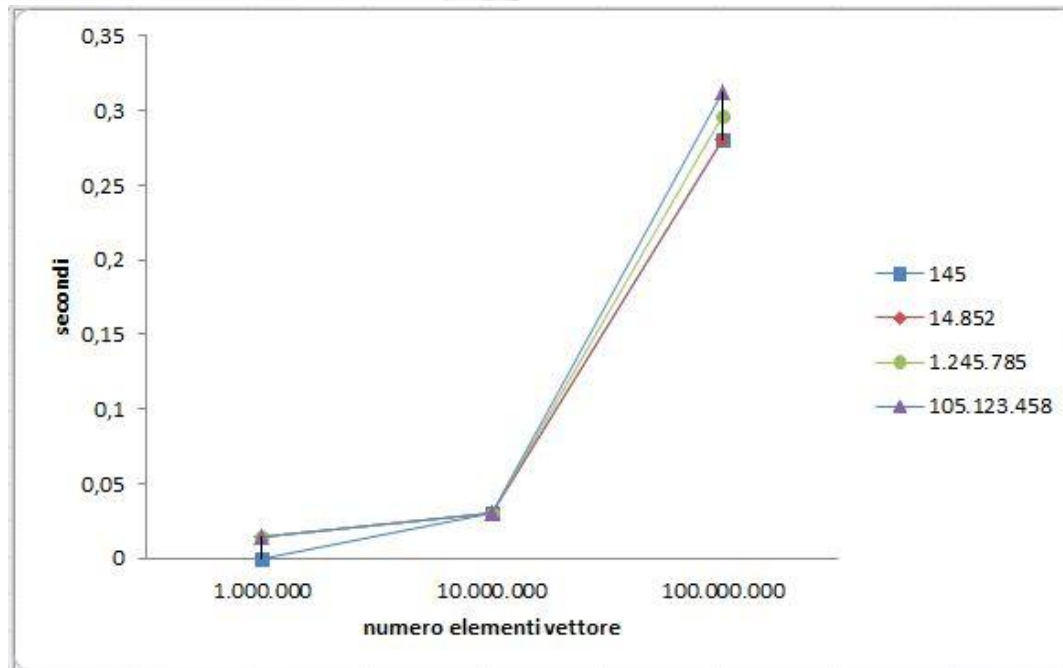
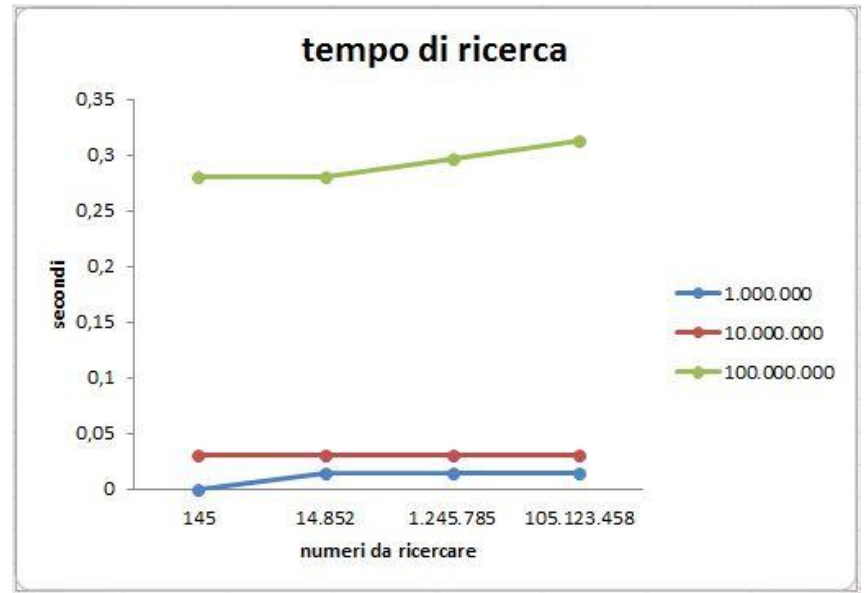
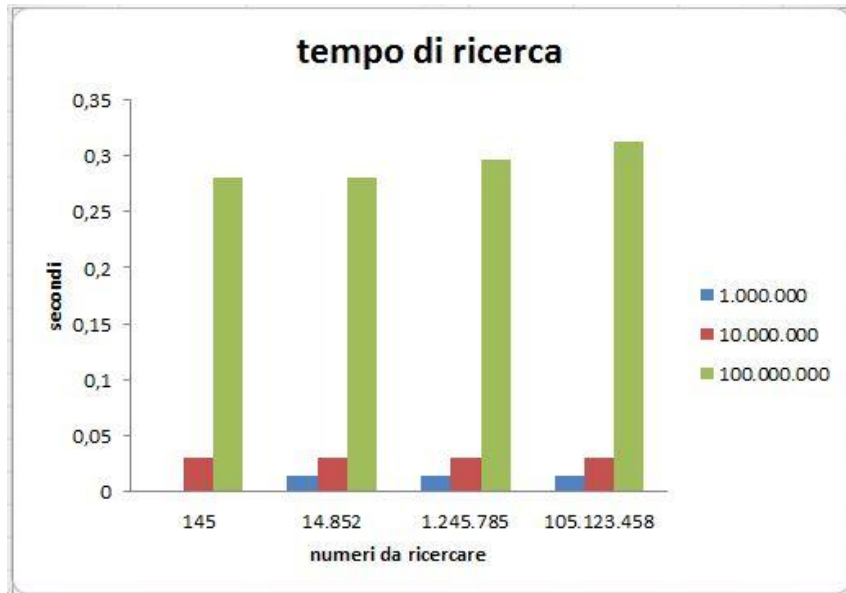
1000000 elementi
elemento da ricercare: 145
l'elemento si trova al 146^ posto
ci ho messo 0 clicks (0.000000 secondi)
Premere un tasto per continuare . . .
```




```
D:\università\informatica\2 anno\algoritmi e strutture dati\ricerca binaria\iterativa\iterativa.exe

1000000 elementi
elemento da ricercare: 145
l'elemento si trova al 146^ posto
ci ho messo 15 clicks (0.015000 secondi)
Premere un tasto per continuare . . .
```

Un po' di grafici



Tempi di esecuzione per un range: 100-100.000

- Non inseriamo più di 100.000 elementi perché l'esecuzione del programma durerebbe troppo tempo.
 - Facciamo 4 serie di prova ognuna che ricerca 4 valori diversi come si vede dalla tabella riassuntiva:
 - I risultati non sono sempre gli stessi
 - Stampando tutti gli elementi non viene calcolato solamente il tempo di esecuzione della funzione ITERATIVA ma anche il tempo che viene perso per stampare l'intero vettore
 - Non è possibile inserire l'inizializzazione del clock prima di richiamare la funzione e la terminazione dopo perché il risultato è sempre 0
- 

| lasciando la stampa degli elementi del vettore | | | | |
|------------------------------------------------|-------|-------|--------|-----------|
| | 145 | 1.235 | 14.852 | 1.245.785 |
| 100 | 0,015 | 0,015 | 0,046 | 0,031 |
| 1000 | 0,078 | 0,062 | 0,156 | 0,14 |
| 10000 | 0,686 | 0,686 | 0,702 | 0,639 |
| 100000 | 6,832 | 7,035 | 6,801 | 6,52 |

```
int main(void)
{
    [...]
    num=145;
    printf("\nelemento da ricercare: %d\n",num);

    t = clock();

    if(ric_b(v,num,n)!=-1) printf("l'elemento si trova al %d^ posto\n",ric_b(v,num,n));
    else printf("elemento non trovato\n");

    t = clock()-t;

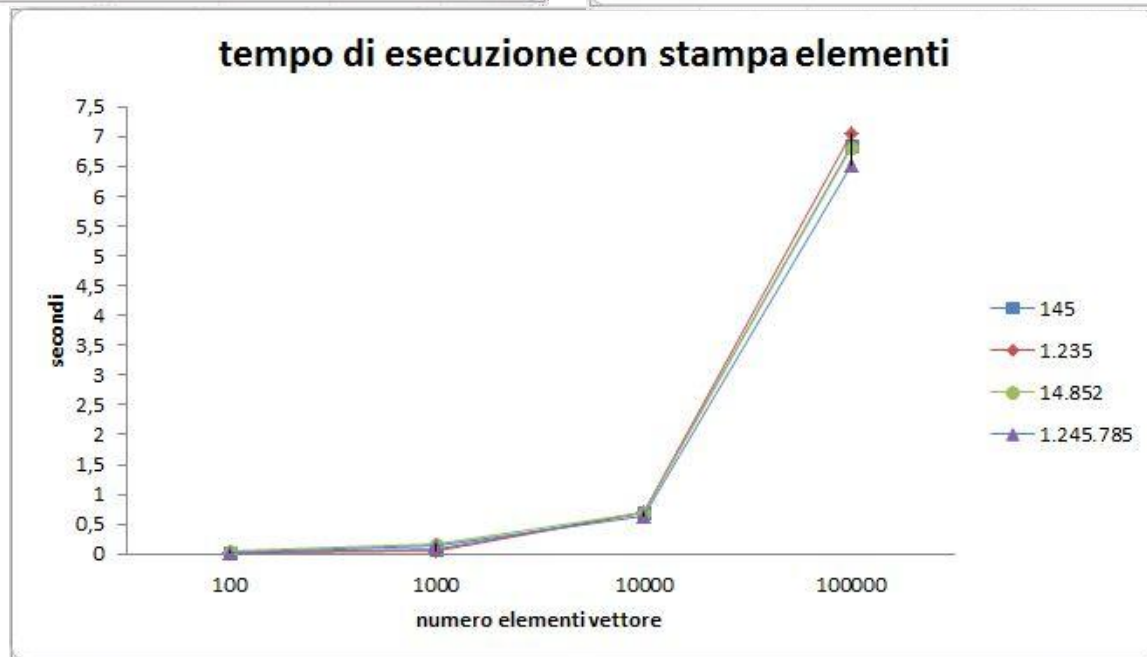
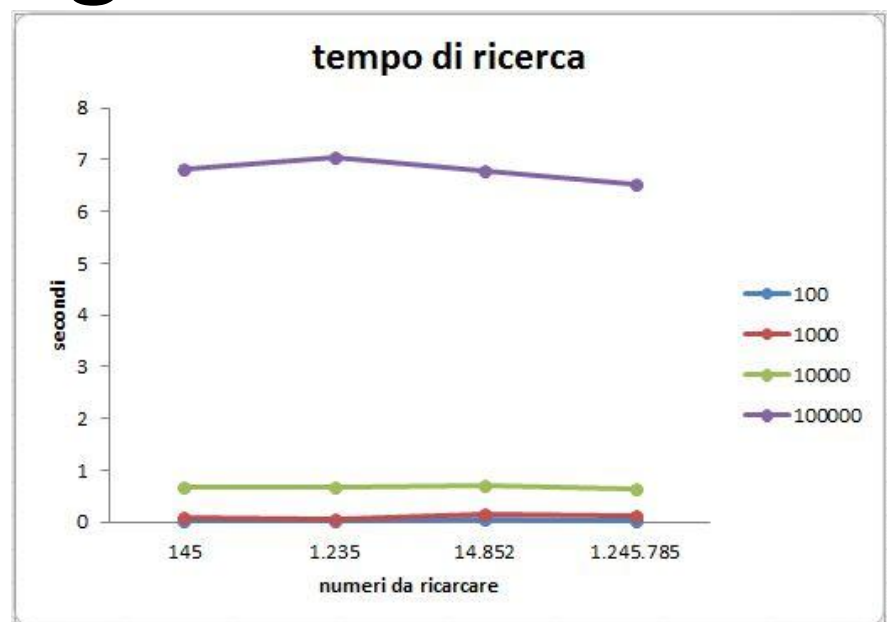
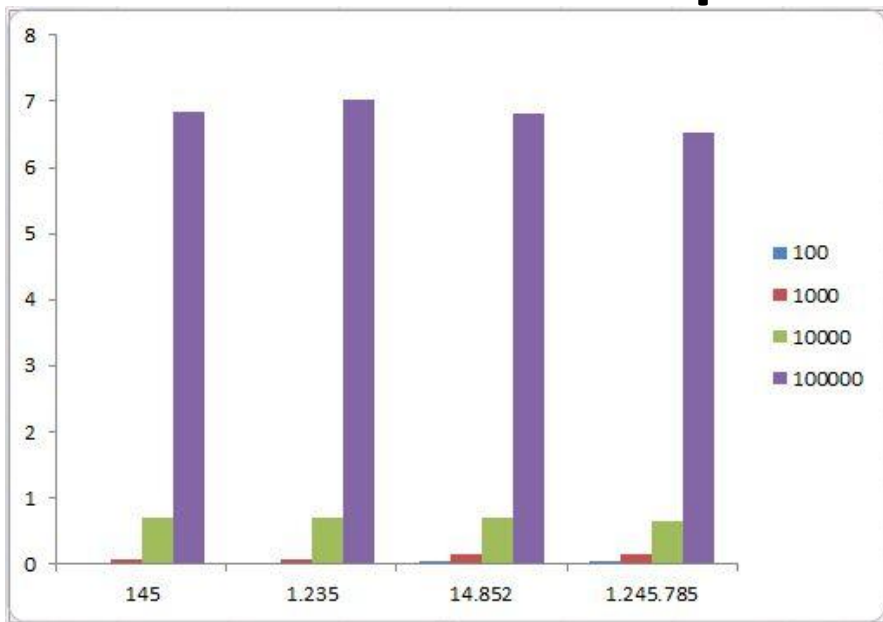
    printf ("\nci ho messo %d clicks (%f secondi)\n",t,((float)t)/CLOCKS_PER_SEC);
    free(v); //deallocazione memoria
    [...]
}
```


I risultati non sono sempre gli stessi:

```
D:\università\informatica\2 anno\algoritmi e strutture dati\ricerca binaria\iterativa\iterativa.exe
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 elementi
elemento da ricercare: 145
elemento non trovato
ci ho messo 31 clicks (0.031000 secondi)
Premere un tasto per continuare . . .
```

```
D:\università\informatica\2
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 elementi
elemento da ricercare: 145
elemento non trovato
ci ho messo 46 clicks (0.046000 secondi)
Premere un tasto per continuare . . .
```

Un po' di grafici:



Ricerca binaria ricorsiva (main)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int ric_b(int *,int,int,int);

int main(void)
{
    int *v,i,n,num;
    clock_t t;

    t = clock(); //inizializzazione clock se viene posizionato
    prima della chiamata della funzione ric_b il clock
    risulterà sempre 0
    n=1000000;

    v=malloc(n*sizeof(int)); //allocazione memoria vettore
    for(i=0; i<n; i++) //assegnazione valore
    {
        v[i]=i;
    }

    /*for(i=0; i<n; i++) //se viene tolta la stampa il clock
    risulterà sempre 0 tranne per un range circa
    1.000.000:100.000.000
    {
        printf("%d\n",v[i]);
    }*/

    printf("\n%d elementi\n",n);
    num=145;
    printf("\nelemento da ricercare %d\n",num);

    // t = clock();

    if(ric_b(v,num,0,n-1)!=-1) printf("l'elemento si trova
    al %d^o posto\n",ric_b(v,num,0,n-1));
    else printf("elemento non trovato\n");

    t = clock() - t;
    printf ("\n ci ho messo %d clicks (%f
    secondi)\n",t,((float)t)/CLOCKS_PER_SEC);
    free(v); //deallocazione memoria

    system("PAUSE");
    return 0;
}
```

Funzione ricorsiva

```
int ric_b(int *vett,int ele,int i,int f)
{
    int m;
    if(i>f) return -1;
    else
    {
        m=(i+f)/2;
        if(vett[m]==ele) return m+1;
        else
            if(ele<vett[m]) return(ric_b(vett,ele,i,m-1));
            else return(ric_b(vett,ele,m+1,f));
    }
}
```

**Tempi di esecuzione per un range:
1.000.000-100.000.000**

- Se inseriamo un numero di elementi inferiore a 1.000.000 il clock restituirà sempre un valore pari a 0.
- Facciamo 3 serie di prove ognuna che ricerca 4 valori diversi come si vede dalla tabella riassuntiva:
- I risultati non sono sempre gli stessi
- Non stampiamo gli elementi del vettore (vedremo che stampandoli i tempi di esecuzione aumenteranno considerevolmente)

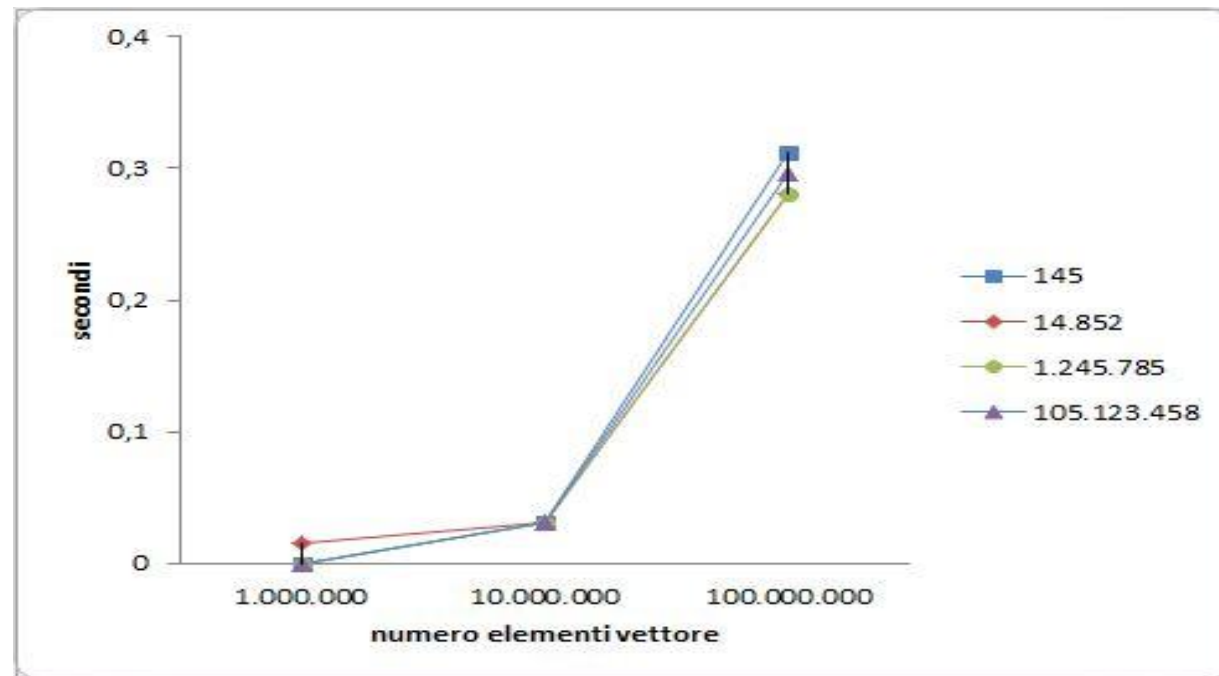
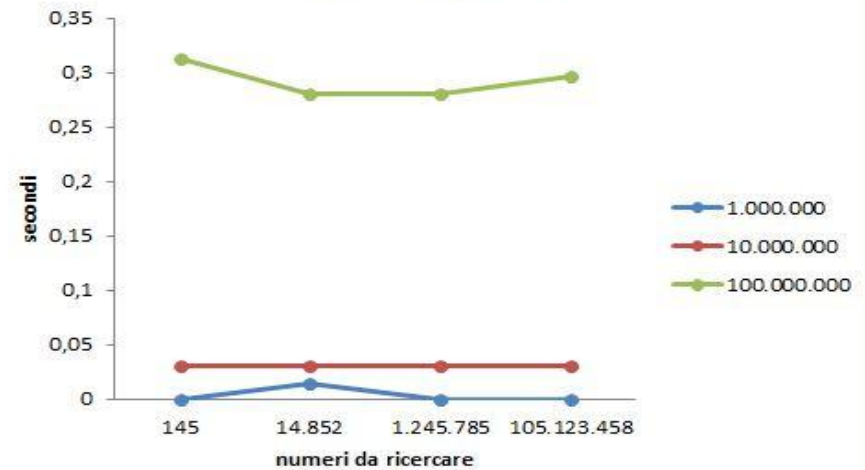
| Ricerca Binaria ricorsiva | | | |
|---------------------------|-------------|-------------|---------------------|
| | n° elementi | 1.000.000 | numero da ricercare |
| | clicks | secondi | |
| 1 | 0 | 0 | 145 |
| 2 | 15 | 0,015 | 14.852 |
| 3 | 0 | 0 | 1.245.785 |
| 4 | 0 | 0 | 105.123.458 |
| | n° elementi | 10.000.000 | |
| 1 | 31 | 0,031 | 145 |
| 2 | 31 | 0,031 | 14.852 |
| 3 | 31 | 0,031 | 1.245.785 |
| 4 | 31 | 0,031 | 105.123.458 |
| | n° elementi | 100.000.000 | |
| 1 | 312 | 0,312 | 145 |
| 2 | 280 | 0,28 | 14.852 |
| 3 | 280 | 0,28 | 1.245.785 |
| 4 | 296 | 0,296 | 105.123.458 |

Un po' di grafici

tempo di ricerca



tempo di ricerca



Tempi di esecuzione per un range: 100-100.000

- Non inseriamo più di 100.000 elementi perché l'esecuzione del programma durerebbe troppo tempo.
- Facciamo 4 serie di prova ognuna che ricerca 4 valori diversi come si vede dalla tabella riassuntiva:
- I risultati non sono sempre gli stessi
- Stampando tutti gli elementi non viene calcolato solamente il tempo di esecuzione della funzione RICORSIVA ma anche il tempo che viene perso per stampare l'intero vettore
- Non è possibile inserire l'inizializzazione del clock prima di richiamare la funzione e la terminazione dopo perché il risultato è sempre 0



| lasciando la stampa degli elementi del vettore | | | | |
|------------------------------------------------|-------|-------|--------|-----------|
| | 145 | 1.235 | 14.852 | 1.245.785 |
| 100 | 0,015 | 0,031 | 0,015 | 0,015 |
| 1000 | 0,078 | 0,078 | 0,093 | 0,171 |
| 10000 | 0,624 | 0,655 | 0,624 | 0,608 |
| 100000 | 6,77 | 6,567 | 6,536 | 6,583 |

```
int main(void)
{
    [...]
    num=145;
    printf("\nelemento da ricercare: %d\n",num);

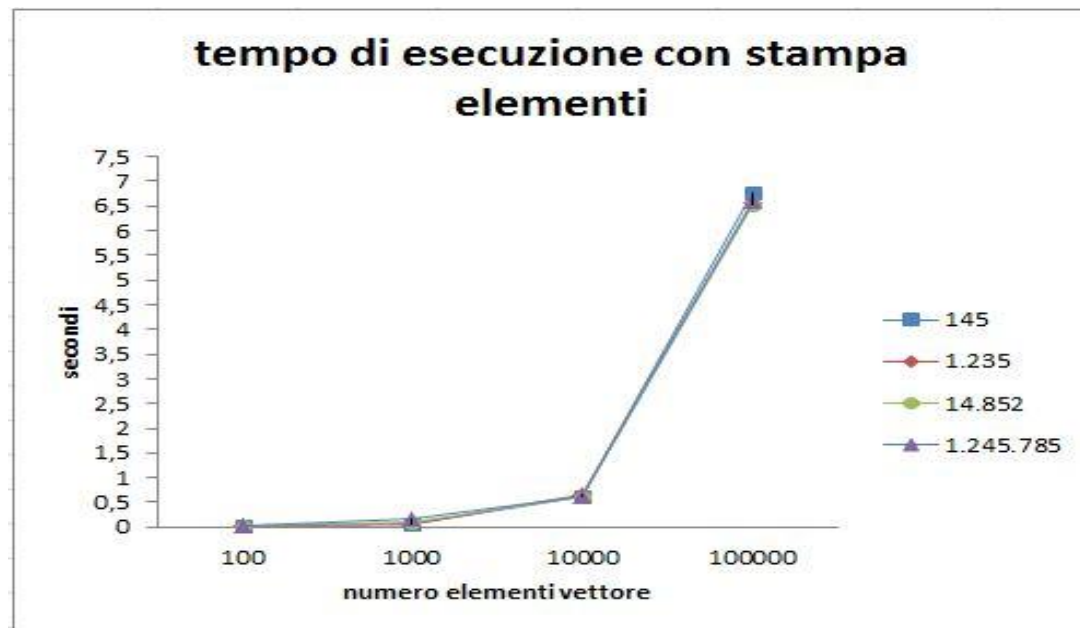
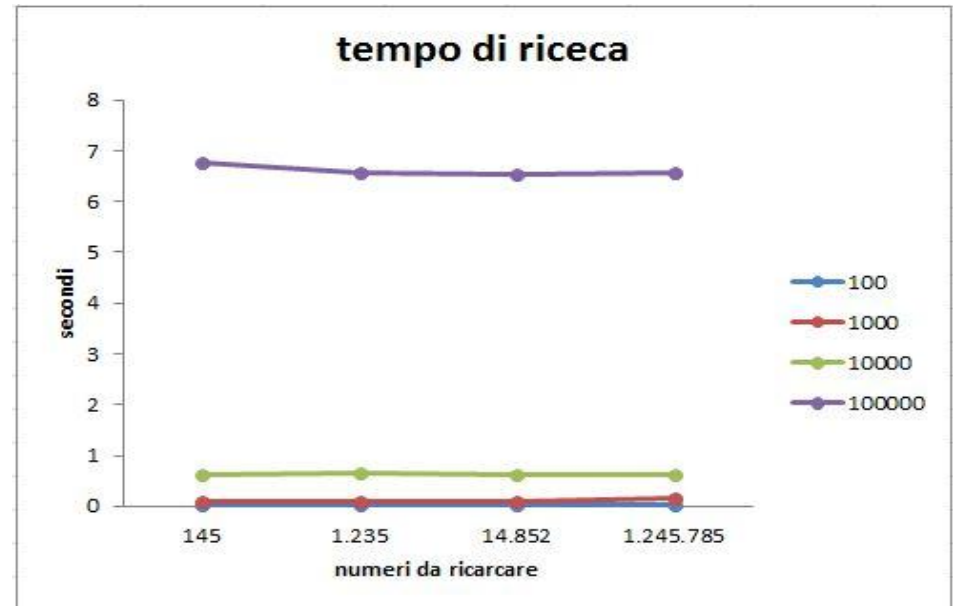
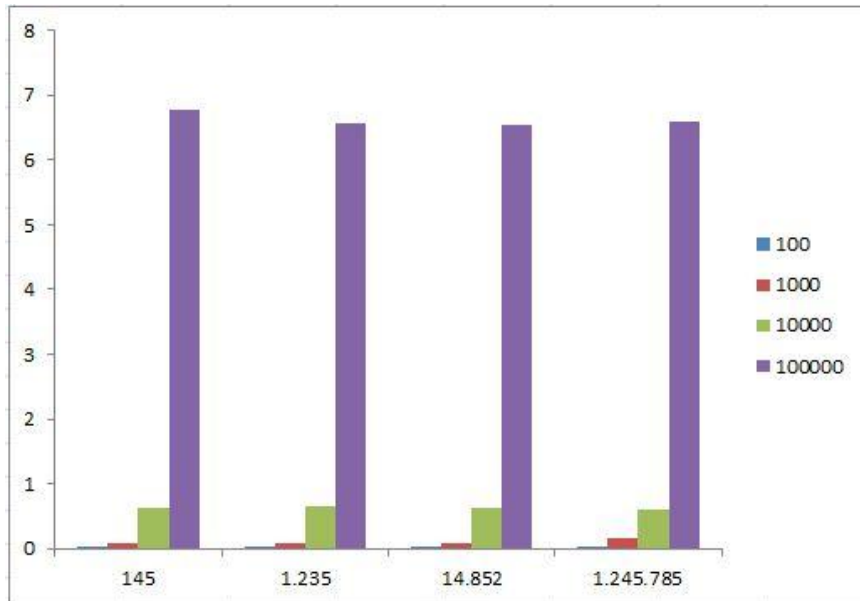
    t = clock();

    if(ric_b(v,num,0,n-1)!=-1) printf("l'elemento si trova al %d^
posto\n",ric_b(v,num,0,n-1));
    else printf("elemento non trovato\n");

    t = clock()-t;

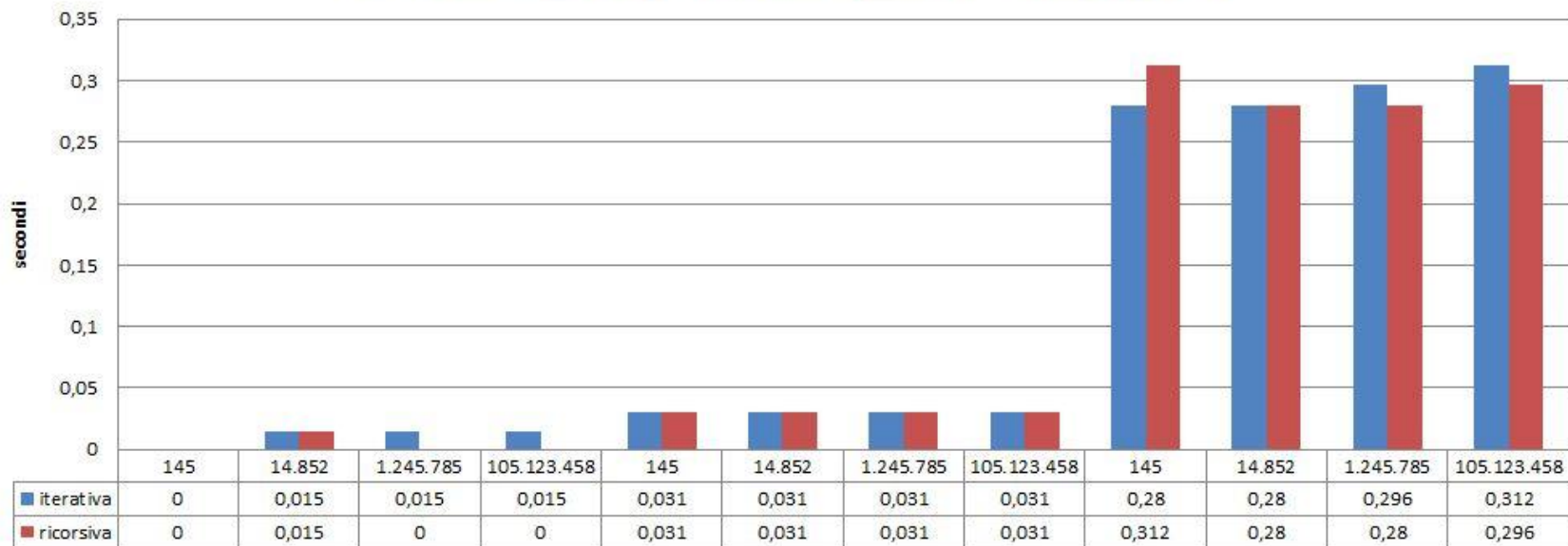
    printf ("\nci ho messo %d clicks (%f secondi)\n",t,((float)t)/CLOCKS_PER_SEC);
    free(v); //deallocazione memoria
    [...]
}
```

Un po' di grafici

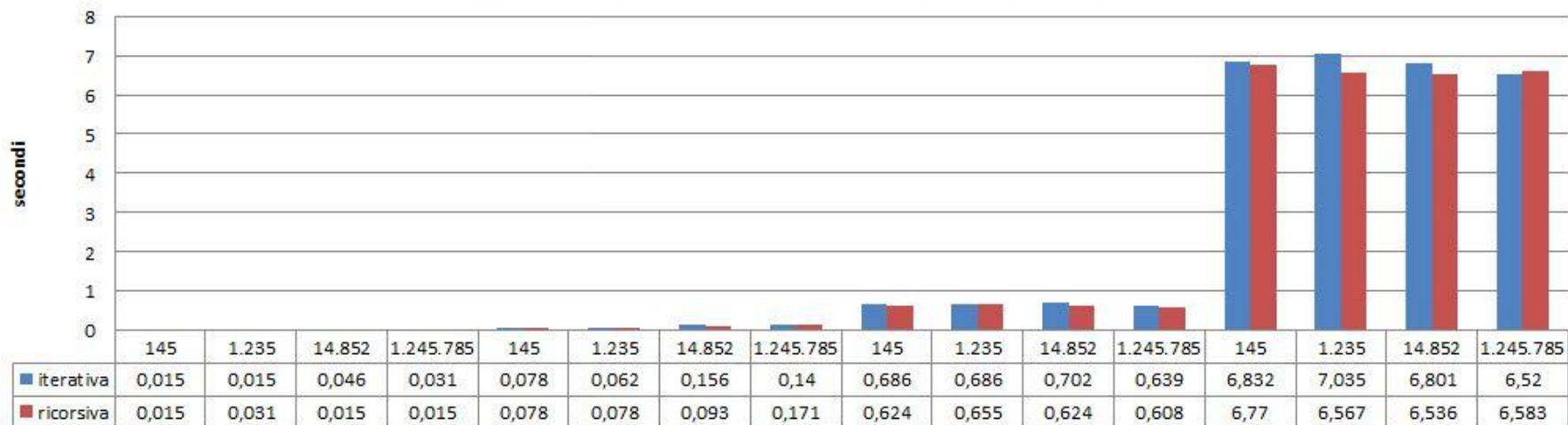


Confronto tra i due modelli

Differenza tempi di esecuzione (1.000.000-100.000.000)



Differenza tempi di esecuzione (100-100.000)



Conclusioni

Le diverse interpretazioni dei dati possono essere:

1. i due algoritmi non si differenziano rispetto ai tempi di esecuzione
2. la funzione `clock()` non è adatta per calcolare il tempo per questi tipi di problemi che hanno esecuzione molto veloce (lo stesso problema si riscontra nell'algoritmo di fibonacci, solo con un input maggiore di 1.000.000 il risultato del `clock` è diverso da 0).