Търсене

Множество от данни наричаме **подредено** (сортирано), ако подреждането на елементите на множеството удовлетворява някаква предварително зададена наредба.

Последователно търсене

```
public int Search(int[] A, int n, int x)
{
  for(int i = 0; i < n; i++)
    if(A[i] == x)
      return 1;
  return 0;
}
```

Сложност при успешно търсене – O(n), при неуспешно – Θ(n)

Търсене със стъпка

Търсенето е в сортирано множество. Избираме стъпка k и сравняваме с x елементите $A[k], A[2k], \ldots, A[mk]$, докато $mk \leq n$

```
public int SearchStep(int[] arr, int n, int x, int k) {
  int i = 0;
  while(i < n && arr[i] < x) {
    i += k;
  }
  if(arr[i] <= x) {
    for(int j = i-k+1; j<=i; j++){
      if(arr[j] == x){
        return 1;
      }
    }
  }
  return 0;
}
```

Сложност при успешно търсене – O(√n), при неуспешно – O(√n)

Двоично търсене

Търсим елемента *x* в сортиран масив. Със *l* и *r* означаваме левия и десния край на масива

1. Определяме елемента *y = A[(l + r)/2] (средния елемент в масива)*
2. Сравняваме *y* и *x*. Ако *x == y* – край (да);
3. Ако x < y, то *r = (l + r − 1)/2*. В противен случай *l = (l + r + 1)/2;*
4. Ако *l != r* – стъпка 1. В противен случай проверяваме дали *A[l] == x*. Ако да – елементът е в масива. В противен случай не е.

```
public int BinarySearch(int[] arr, int n, int x) {
  int l = 0, r = n-1;
  while(l != r) {         //Продължи докато левия и десния край се срещнат
    int y = arr[(l+r)/2]; //y = Средния елемент
    if (x == y) return 1; //Намерихме го
    else {
      // Ако x < y десния край става средата на текущия отрез
      if(x < y) {
        r = (l + r - 1)/2;
      // Иначе левия край става средата на текущия отрез
      } else {
        l = (l + r + 1)/2;
      }
    }
  if(arr[l] == x) return 1;
  return 0;
}
```

Сложност при търсене – $O(\log_2 n)$

Фибоначиево търсене

Наподобява двоичното търсене – масивът се разделя на две части, от които едната се отхвърля. Разликата е в елемента, който избираме на всяка стъпка. Ако масивът е с дължина Fk − 1, то на всяка стъпка избираме y = A[Fk−1]. Ако x < y, търсим измежду елементите с индекси *1, 2, . . . , Fk−1 −1*. Ако x > y, търсим измежду елементите с индекси *Fk−1 + 1, . . . , Fk − 1*.

На практика, фибоначиевото търсене строи балансирано двоично дърво, известно като *Фибоначиево дърво*. Дълбочината на това дърво е с 45 % поголяма от тази на дърветата, които се изграждат при двоичното търсене ⇒ **можем да очакваме 45 % забавяне спрямо двоичното търсене.**