

Теоретический материал

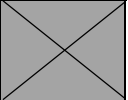
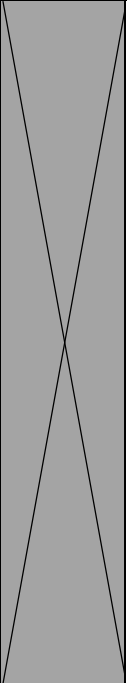
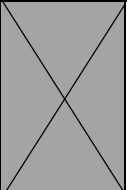
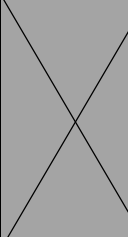
Двоичный (бинарный) поиск (также известен как метод деления пополам или дихотомия) — классический алгоритм поиска элемента в отсортированном массиве, использующий дробление массива на половины.

Алгоритм включает в себя следующие шаги:

1. Определение значения элемента в середине массива. Полученное значение сравнивается с искомым элементом.
2. Если искомый элемент меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.
3. Поиск сводится к тому, что вновь определяется значение срединного элемента в выбранной половине и сравнивается с искомым элементом.
4. Процесс продолжается до тех пор, пока не будет найден элемент, равный искомому, или не станет пустым интервал для поиска.

Существует ряд особенностей, которые необходимо учитывать при программной реализации кода:

1. Пусть `first` и `last` – индексы элементов, являющихся левым и правым концом отрезка поиска на некотором шаге алгоритма. В случае больших массивов код $(first + last) / 2$ для поиска середины отрезка может быть ошибочен, если `first` и `last` по отдельности умещаются в свой тип, а `first+last` — нет. Для обхода этого ограничения можно использовать выражение $first + (last - first) / 2$, которое приведет к такому же результату, но позволит не выйти за границы типа данных.
2. Необходимо тестировать код для случаев пустого массива. Массива из одного элемента и другие частные случаи в зависимости от условий задачи.
3. Массив может содержать несколько экземпляров искомого элемента. В зависимости от условия задачи, необходимо учитывать требования поиска первого экземпляра элемента, последнего экземпляра элемента, произвольного экземпляра и т.п.

Пример 1.1	
Задача:	
	Написать программу для бинарного поиска элемента в отсортированном массиве
Решение:	
	<pre> 1 def binary_search(list, key): 2 low = 0 3 high = len(list) - 1 4 5 while low <= high: 6 mid = (low + high) // 2 7 midVal = list[mid] 8 if midVal == key: 9 return mid 10 if midVal > key: 11 high = mid - 1 12 else: 13 low = mid + 1 14 return 'Элемент не найден' 15 16 a = [1, 2, 3, 4, 5, 6, 7] 17 print(binary_search(a,3)) </pre>
Ответ:	
	<pre> 2 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Задание 1.1	
Задача:	
	Написать программу для бинарного поиска элемента в отсортированном массиве. В случае, если элемент не найден, указать позицию, в которую можно выполнить вставку элемента (чтобы массив оставался отсортированным) и выполнить вставку
Решение:	

```

1.2 > 1.1 задача.py > ...
1  def binary_search(list, key):
2      low = 0
3      high = len(list) - 1
4
5      while low <= high:
6          mid = (low + high) // 2
7          midVal = list[mid]
8          if midVal == key:
9              return mid
10         if midVal > key:
11             high = mid - 1
12         else:
13             low = mid + 1
14     list.insert(low, key)
15
16     return low, list
17
18 a = [1,2,3,4,5,6,8]
19 print(binary_search(a, 7))
20
21 a = [1,2,3,4,5,6,7,8]
22 print(binary_search(a, 7))
23

```

Ответ:

```

(6, [1, 2, 3, 4, 5, 6, 7, 8])
6

```

Задание 1.2

Задача:

Массив `arr` называется горным, если выполняются следующие свойства:

- 1) В массиве не менее трех элементов
- 2) Существуют i ($0 < i < \text{arr.length} - 1$) что:
 - $\text{arr}[0] < \text{arr}[1] < \dots < \text{arr}[i - 1] < \text{arr}[i]$
 - $\text{arr}[i] > \text{arr}[i + 1] > \dots > \text{arr}[\text{arr.length} - 1]$

Иными словами, в массиве есть пик (или несколько пиков) такие, что остальные элементы убывают влево и вправо относительно пика.

Используя алгоритм бинарного поиска, проверить, является ли массив `arr` горным. Если да – вернуть индекс первого пика.

Решение:

```
1.2 > 1.2 задача.py > ...
1 def gor(list):
2     if len(list) < 3: return "Не горный"
3     else:
4         left = 0
5         right = len(list) - 1
6
7         while left <= right:
8             mid = (left + right) // 2
9             if mid == len(list)-1:
10                break
11             elif list[mid] > list[mid + 1]: right = mid - 1
12             else: left = mid + 1
13         if mid == 0: return "Не горный"
14         elif mid == len(list)-1: return "Не горный"
15         return mid
16
17 print(gor([1,2,3,4,5,6]))
18 print(gor([6,5,4,3,2,1]))
19 print(gor([2,2,2,2,2,2]))
20 print(gor([1,2,3,4,2,1]))
21
```

Ответ:

```
Не горный
Не горный
Не горный
3
```

Задание 1.3

Задача:

Массив отсортирован по возрастанию (элементы могут повторяться). С помощью алгоритма бинарного поиска выяснить, каких чисел в массиве больше – положительных или отрицательных. Если больше положительных чисел – вывести их количество, если же больше отрицательных, то вывести их количество

Решение:

```
1.2 > 1.3 задача.py > ...
1 def binary_search2(list):
2     key = 0
3     low = 0
4     high = len(list) - 1
5     while 0 in list:
6         list.remove(0)
7     while low <= high:
8         mid = (low + high) // 2
9         midVal = list[mid]
10        if midVal == key:
11            return max(mid, len(list) - 1 - mid)
12        if midVal > key:
13            high = mid - 1
14        else:
15            low = mid + 1
16        list.insert(low, key)
17
18    return max(low, len(list) - 1 - low)
19
20 a = [-5, -4, 1, 5, 6, 7, 8]
21 print(binary_search2(a))
22
23 a = [-6, -5, -4, -3, -2, -1, 2, 3, 4]
24 print(binary_search2(a))
25
26 a = [-5, 0, 0, 0, 1, 2, 3, 4]
27 print(binary_search2(a))
```

Ответ:

5
6
4

Задание 1.4*

Задача:

Дан целочисленный массив `nums`. Постройте целочисленный массив `counts`, где `counts[i]` — количество меньших элементов справа от `nums[i]`.

Пример:

Ввод: `nums = [5,2,6,1]`

Вывод: `[2,1,1,0]`

Объяснение:

Справа от 5 находятся 2 меньших элемента (2 и 1).

Справа от 2 находится только 1 элемент меньшего размера (1).

Справа от 6 находится 1 элемент поменьше (1).

Справа от 1 находится 0 элементов меньшего размера.

Решение:

```
1.2 > 1.4 задача.ру > ...
1  def magic(list):
2      itog = [0]*len(list)
3      for i in range(len(list)):
4          for j in range(i+1, len(list)):
5              if list[i] > list[j]:
6                  itog[i] += 1
7      return itog
8
9  print(magic([5,2,6,1]))
10 print(magic([5,5,5,5]))
11 print(magic([1,2,3,4]))
```

Ответ:

```
[2, 1, 1, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
```