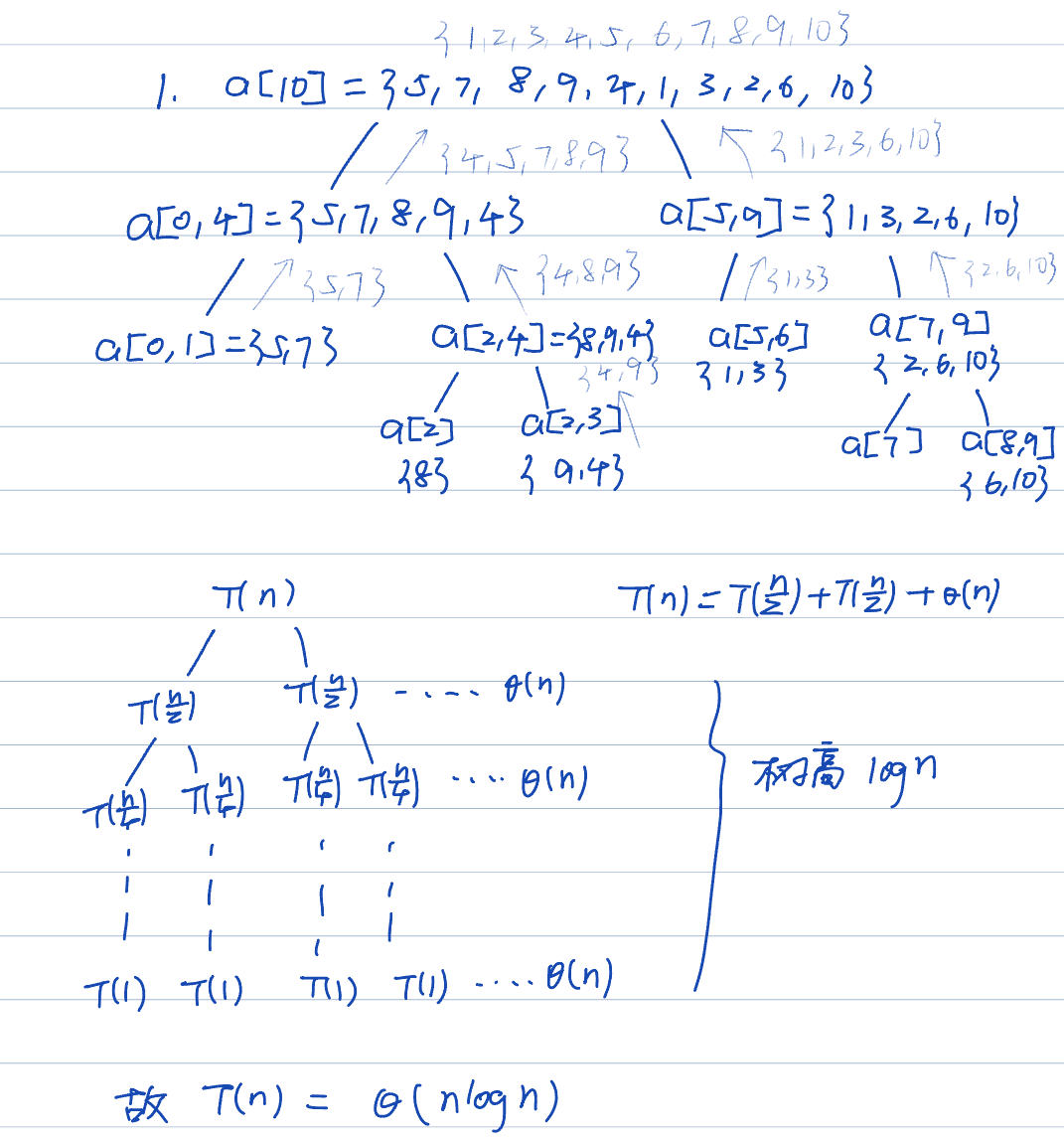
**算法设计与分析第三章作业**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **王靳** | **班级** | **计科十班** | **学号** | 220111012 |
| **第1题** |  | | | | |
| **第2题** |  | | | | |
| **第3题** |  | | | | |
| **第4题** |  | | | | |
| **第5题** |  | | | | |
| **总分** |  | | | | |
| **备注** | 作业提交截止时间：2023-10-13日24:00，超过提交截至时间的作业视为无效。作业提交邮箱：[hitsz\_algorithm@126.com。](mailto:hitsz_algorithm@126.com。)作业文件名命名方式： 第x章-x班-姓名-学号（例， 第1章-1班-张三-220110101.docx）； 邮件主题为：第x章作业, x班，姓名，学号（例， 第1章作业，1班，张三，220110101）。缺少这些信息的作业将被酌情扣分。 | | | | |

1. 将数组进行归并排序，并通过递归树方法计算归并排序时间复杂度。**（该题考察归并排序以及时间复杂度，15分）**

****

1. 存储给定一个个元素有序的（升序）整型数组和一个目标值，写一个时间复杂度为的伪代码搜索中的**（需附完整Python源代码）**，如果目标值存在返回下标，否则返回。证明代码时间复杂度为。**（该题考察分治法以及时间复杂度，15分）**

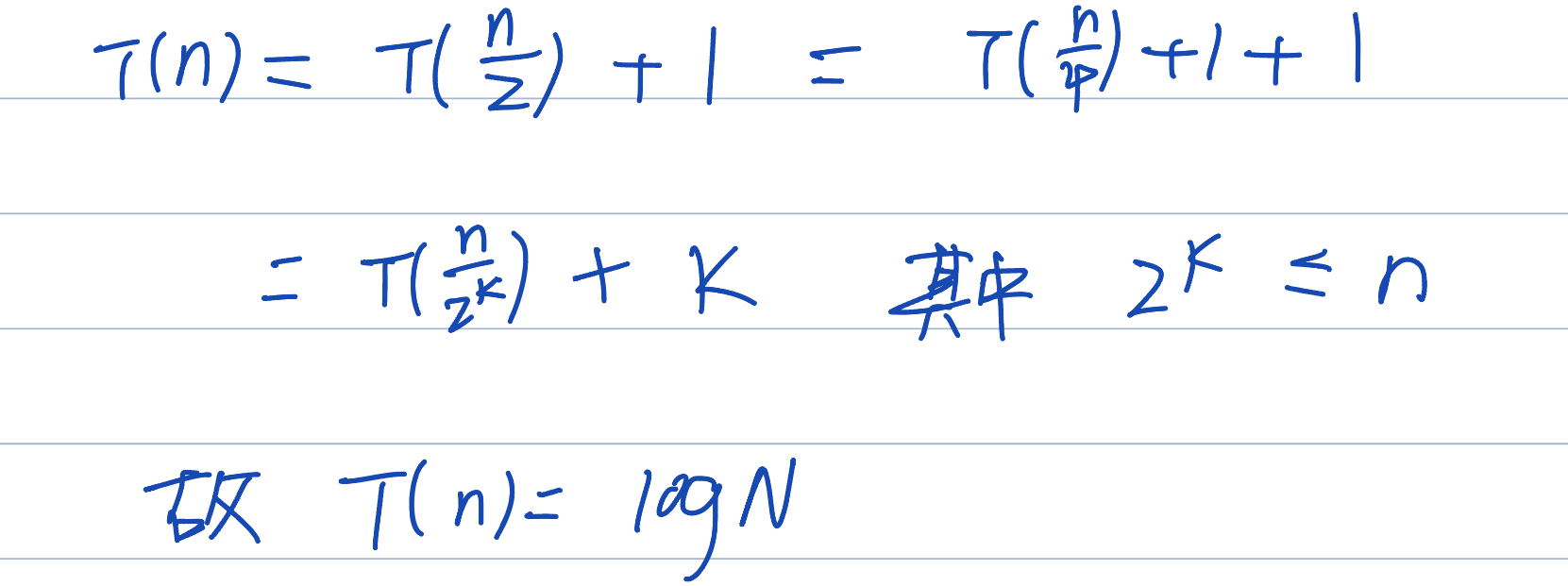
**示例 1: 示例 2:**

**输入: nums =** **[-1,0,3,5,9,12], target = 9 输入: nums = [-****1,0,3,5,9,12], target = 2**

**输出: 4 输出: -1**

**解释: 9 出现在 nums 中并且下标为 4 解释: 2 不存在 nums 中因此返回 -1**

1. def bin\_search\_utils(arr, l, r, target):
2. *# base case*
3. if l == r:
4. return l
5. elif l > r:
6. return -1
7. mid = l + ((r - l) >> 2)
8. if arr[mid] > target:
9. *# 在左边找*
10. return bin\_search\_utils(arr, l, mid - 1, target)
11. elif arr[mid] < target:
12. return bin\_search\_utils(arr, mid + 1, r, target)
13. else:
14. *# 如果相等，剪枝*
15. return mid
16. def bin\_search(arr, target):
17. *# 防御*
18. if arr == None or len(arr) == 0:
19. return -1
20. return bin\_search\_utils(arr, 0, len(arr) - 1, target)
21. if \_\_name\_\_ == '\_\_main\_\_':
22. arr = [-1, 0, 3, 5, 9, 12]
23. index = bin\_search(arr, 9)
24. print(index)
25. index = bin\_search(arr, 2)
26. print(index)



1. 给定由个互不相同的数组成的集合以及正整数，试设计一个时间复杂度的算法，并写出伪代码，找出中最接近的中位数的个数。**（该题考察分治策略，20分）**
2. def bfprt(arr, k):
3. if len(arr) == 1:
4. return arr[0]
5. pivot = select\_pivot(arr)
6. left = [x for x in arr if x < pivot]
7. right = [x for x in arr if x > pivot]
8. equal = [x for x in arr if x == pivot]
9. if k <= len(left):
10. return bfprt(left, k)
11. elif k > len(left) and k <= len(left) + len(equal):
12. return equal[0]
13. else:
14. return bfprt(right, k - len(left) - len(equal))
15. def select\_pivot(arr):
16. if len(arr) <= 5:  *# 中间位置*
17. return sorted(arr)[len(arr) >> 1]
18. chunks = chunked(arr, 5)
19. full\_chunks = [chunk for chunk in chunks if len(chunk) == 5]  *# 完整的chunk*
20. sorted\_groups = [sorted(chunk) for chunk in full\_chunks]  *# 给长度为5的chunk排序*
21. medians = [chunk[2] for chunk in sorted\_groups]  *# 这个是中位数的集合*
22. median\_of\_medians = bfprt(medians, len(medians) >> 1)  *# 中之王者*
23. return median\_of\_medians
24. def chunked(iterable, n):  *# 将n个打包成一个数组，在放到一个二维数组中*
25. return [iterable[i:i + n] for i in range(0, len(iterable), n)]
26. def middle\_k(arr, k):
27. n = len(arr)
28. n\_div\_2 = n >> 1
29. mid = bfprt(arr, n\_div\_2)
30. print(mid)
31. bucket = []
32. ans = []
33. for i in range(n):
34. bucket.append(abs(arr[i] - mid))
35. ret = bfprt(bucket, k)
36. for i in range(n):
37. temp = abs(arr[i] - mid)
38. if temp <= ret:
39. ans.append(arr[i])
40. return ans
41. if \_\_name\_\_ == "\_\_main\_\_":
42. *# 示例*
43. arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
44. ans = middle\_k(arr, 3)
45. print(ans)
46. 输入两个大小分别为和的正序（从小到大）数组和。设计算法，并编写时间复杂度为的伪代码**（需附完整Python源代码）**，找出并返回这两个正序数组的中位数。**（该题考察分治法以及时间复杂度，20分）**

**示例 1： 示例2：**

**输入：nums1=[1,3], nums2=[2] 输入：nums1=[1,2], nums2=[3,4]**

**输出：2.00000 输出：2.50000**

**解释：合并数组=[1,2,3]，中位数2 解释：合并数组=[1,2,3,4]，中位数 (2+3)/2=2.5**

1. *# 找两个数组第k小的，m+n是奇数*
2. def find\_k\_smallest(arr1, arr2, k):
3. if not arr1:
4. return arr2[k - 1]
5. if not arr2:
6. return arr1[k - 1]
7. if len(arr1) > len(arr2):
8. arr1, arr2 = arr2, arr1
9. if k == 1:
10. return min(arr1[0], arr2[0])
11. *# 查找第k/2小的数*
12. i = min(len(arr1), k >> 1)
13. j = min(len(arr2), k >> 1)
14. if arr1[i - 1] < arr2[j - 1]:
15. arr1 = arr1[i:]
16. return find\_k\_smallest(arr1, arr2, k - i)
17. else:
18. arr2 = arr2[j:]
19. return find\_k\_smallest(arr1, arr2, k - j)
20. def find\_mid(arr1, arr2):
21. m = len(arr1)
22. n = len(arr2)
23. if (((m + n) >> 1) << 1) == (m + n):
24. *# 如果是偶数*
25. mid\_low = find\_k\_smallest(arr1, arr2, (m + n) >> 1)
26. mid\_high = find\_k\_smallest(arr1, arr2, ((m + n) >> 1) + 1)
27. return (mid\_low + mid\_high) / 2
28. else:
29. *# 如果是奇数*
30. mid\_high = find\_k\_smallest(arr1, arr2, ((m + n) >> 1) + 1)
31. return mid\_high
32. if \_\_name\_\_ == "\_\_main\_\_":
33. arr1 = [1, 2, 5]
34. arr2 = [3, 4, 6]
35. print(find\_mid(arr1, arr2))
36. 输入一个整数数组和一个整数，请输出其中出现频率前高的元素，设计算法并写出伪代码**（需附完整Python源代码）**。**（该题考察分治法中快速排序划分的思想，30分）**

**示例 1: 示例 2:**

**输入: nums = [****1,1,1,2,2,3], k = 2 输入: nums = [1], k = 1**

**输出: [1,2] 输出: [1]**

一种想法：bfprt算法，找到第k大的频数k\_f，然后找到频数>=k\_f的元素，然后得到一个列表，然后在（关键字-频数）字典中找到对应的关键字。

1. def bfprt(arr, k):
2. if len(arr) == 1:
3. return arr[0]
4. pivot = select\_pivot(arr)
5. left = [x for x in arr if x < pivot]
6. right = [x for x in arr if x > pivot]
7. equal = [x for x in arr if x == pivot]
8. if k <= len(left):
9. return bfprt(left, k)
10. elif k > len(left) and k <= len(left) + len(equal):
11. return equal[0]
12. else:
13. return bfprt(right, k - len(left) - len(equal))
14. def select\_pivot(arr):
15. if len(arr) <= 5:  *# 中间位置*
16. return sorted(arr)[len(arr) >> 1]
17. chunks = chunked(arr, 5)
18. full\_chunks = [chunk for chunk in chunks if len(chunk) == 5]  *# 完整的chunk*
19. sorted\_groups = [sorted(chunk) for chunk in full\_chunks]  *# 给长度为5的chunk排序*
20. medians = [chunk[2] for chunk in sorted\_groups]  *# 这个是中位数的集合*
21. median\_of\_medians = bfprt(medians, len(medians) >> 2)  *# 中之王者*
22. return median\_of\_medians
23. def chunked(iterable, n):  *# 将n个打包成一个数组，在放到一个二维数组中*
24. return [iterable[i:i + n] for i in range(0, len(iterable), n)]
25. def top\_k\_frequent\_elements(arr, k):
26. *# 频数*
27. frequency = {}
28. for num in arr:
29. if num in frequency:
30. frequency[num] += 1
31. else:
32. frequency[num] = 1
33. fre\_value = frequency.values()
34. kth\_value = bfprt(arr, len(fre\_value) - k + 1)
35. *# 划分*
36. ans\_value = [value for value in fre\_value if value >= kth\_value]
37. ans\_key = []
38. for key, value in frequency.items():
39. if value in ans\_value:
40. ans\_key.append(key)
41. return ans\_key
42. if \_\_name\_\_ == "\_\_main\_\_":
43. arr = [1, 1, 2, 2, 3, 4, 4, 4, 5, 5]
44. ans = top\_k\_frequent\_elements(arr, 2)
45. print(ans)

一种想法：快速排序。就是快速排序划分。

传入的二元谓词是return a.freq < b.freq

1. class Solution:
2. def topKFrequent(self, nums: List[int], k: int) -> List[int]:
3. n = len(nums)
4. num\_freq = collections.Counter(nums)
5. self.res = []
6. xfs = list(num\_freq.items())
7. xfn = len(xfs)
8. self.quick\_sort(xfs, 0, xfn - 1, k)
9. return self.res
11. def quick\_sort(self, xfs, l: int, r: int, k: int) -> None:
12. pivot\_xf = xfs[l]
13. pivot\_f = xfs[l][1]
14. idx = l
15. for i in range(l + 1, r + 1, 1):
16. if xfs[i][1] >= pivot\_f:
17. idx += 1
18. xfs[idx], xfs[i] = xfs[i], xfs[idx]
19. xfs[l], xfs[idx] = xfs[idx], xfs[l]
20. if k <= idx - l + 1:
21. for i in range(l, idx + 1):
22. self.res.append(xfs[i][0])
23. elif k < idx - l + 1:
24. self.quick\_sort(xfs, l, idx - 1, k)
25. else:
26. for i in range(l, idx + 1):
27. self.res.append(xfs[i][0])
28. self.quick\_sort(xfs, idx + 1, r, k - (idx - l + 1))