Quicksort：快速排序，参数为 一个array，挑一个数pivot做为中间数（一般为第一个或者随机挑选）把数组分为两部分进行递归排序，然后将排好的左树右树和pivot结合起来

选择第一个：

空间消耗：可以是O（n）或O（n2）

Best case：O（nlog(n)）

不断挑到中间的数，形成均衡二叉树

Average： O（nlog(n)）

Worst case：O（n2）

不断挑到最大或最小，形成单链的二叉树

是stable的，不是尾递归

选随机：

空间消耗：可以是O（n）或O（n2）

Best case：O（nlog(n)）

不断挑到中间的数，形成均衡二叉树

Average（期望）： O（nlog(n)）

随机选择是绝对多数贴近Average

Worst case：O（n2）

不断挑到最大或最小，形成单链的二叉树

不是stable的，不是尾递归

Quickselect：快速选择，参数为一个array，一个i（表示寻找第ith最小）挑一个数作为pivot作为中间数（一般为第一个或者随机挑选）把数组分为两部分，只进入一边进行搜索（i小于左边数目搜索左边，否则搜索右边）

第一个或者随机

空间消耗：可以是O（n）或O（n2）

Best case：O（1） 第一个pivot就是要选的那个

Average（期望时间）： O（n）

随机选择大多都接近average

Worst case：O（n2）

不断挑到最大或最小，形成单链的二叉树

是尾递归

Mergesort：合并排序，从中间把数组分成两部分，进行递归排序，把返回的已排序晚的数组进行合并。两个指针指向两个数组的开头，不断往后比较，较小的放进新数组。

空间消耗：O（n2）

时间消耗：O（nlog(n)）

是stable的，不是尾递归

stable sort：相同的数在原数组的顺序，在排序后仍然是一样的顺序

divide-n-conquer: 分治算法

number of inversions：逆序数计算。基于合并排序，在合并时，加上逆序的个数

空间消耗：O（n2）

时间消耗：O（nlog(n)）

是stable的，不是尾递归

longest path in binary tree：找二叉树里最长的路径。按深度优先遍历找下去。返回两个参数，一个是以这个节点为root的树的深度，一个是以这个节点为root的树的最长路径数。

空间消耗：O（n）

时间消耗：O（n）

不是尾递归

k numbers closest to input query, unsorted： b [ai  - x], num = qselect(k, b[] ), 扫描一遍数组b，输出小于num的下表index的a[index]

空间消耗：O（n）

时间消耗：O（n）

不是尾递归

qsort with randomized pivot can be made stable by 3-way partition：

balance tree: for each node. abs(hight(left) – height(right)) <= 1

不断的插入数保持平衡 时间消耗（log（n））

Heaps

priority Queue

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Insert | Pop-min | Peak | Decrease-key | heapify |
| Sorted array | O(n) | O(n) | O(1) | O(n) | O(nlogn) |
| Unsorted array | Amortized O(1) | O(n) | O(n) | O(n) | O(1) |
| heap | O(logn) | O(logn) | O(1) | O(logn) | O(n) |
| Sorted Linklist | O(n) | O(1) | O(1) |  | O(nlogn) |
| Rev-sorted array | O(n) | O(1) | O(1) |  | O(nlogn) |
|  |  |  |  |  |  |

HW4 – 3时间复杂度：O(n \* logk)

HW4 – 2时间复杂度：O(n \* logk)

def qsort(a):

if a == []:

return []

index = randint(0,len(a) - 1)

a[0], a[index] = a[index], a[0]

pivot = a[0]

left = [x for x in a[1:] if x <= pivot]

right = [x for x in a[:] if x > pivot]

return qsort(left) + [pivot] + qsort(right)

datastream: 先heappush k个， 再heapreplace 数字

nbestC midterm 老师的solution (2.8视频后面)

hypergraph

|  |  |  |
| --- | --- | --- |
|  | optimization | Summary |
| graph | MIS | Fib  bitstrings |
| hypergraph | Matrix-chain | # of BST |

HW4 HW5 HW6 期中考会考到，特别HW5老师的solution

0 – 1

opt[i][x] = max(opt[i - 1][x], opt[i - 1][x - wi] + Vi)

unbounded

opt[x] = max (opt[x - wi] + Vi)

i = 1 ~ n

x >= wi

bounded. 期中考也会有类似的

opt[i][x] = max (opt[i - 1][x – j \* Wi]) + j \* Vi

j = 0 ~ min [Ci, X / Wi]

Space Complexity: V = O(nw)

Time Complexity: E = O(nw \* max(Ci))

knapsack (1,sub problem 2, recursion 3, base case )

背包的图

试着用图 看所有问题

midterm, how many edge(Time complexity) how many nodes(Space complexity)

Top-Down(avoids useless case)

heapdict

Dijkstra

(V + E)logV

blackpoint: poped

gray: node in priority q

white: nodes have not been expend

TSP:

opt[s][i] = best path from 0 to i, having visited all cities in S

|  |  |  |  |
| --- | --- | --- | --- |
| Dijkstra | Time compl | E~O(V\*V) | E~O(V) |
| heapdict | (V + E)logV | V\*V\*logV | VlogV |
| Hash(unsorted queue) | V\*V + E | V\*V | V\*V |
| Unsorted/sorted array | V\*V + E \* V | O(V\*V\*V) | O(V\*V) |
| Sorted array in linklist | V + E \* V | O(V\*V\*V) | O(V\*V) |
| Heap | ElogE |  |  |

TSP:

min(opt[1,2,3,4…,n-1][i] + c(i,0)). i = 1 ~ (n - 1)

opt[s][i]（） = min(opt[s – {j}][ j ] + c(j,i)). j 属于 S – {i} ()

Space:

HW8 HW9 Final

RNA:

subproblem: opt[i, j] = max(

opt[i + 1][j – 1] + 1 (if (i, j - 1) pair)

opt[i, k] + opt[k, j] i < k <j

)

TC： O(n \* n \* n)

SP : O(n \* n)

RNA

( + ) (\*)

summation extension

1: 1 - best structure O(n \* n \* n). max +

2: # of structure +. \*

3: k – best : SC O(n \* n) -> O(n\*n\*k).

TC: O(n \* n \* n) O((n \* n) \* ( n + klogn)) = O((n\* n \* n) + (n \* n \* klogn)

opt [i, j] = (+)