第 3 次课堂作业

股票买卖：
解法：

```python
def findbest(input):
    # best_couple = ""
    if len(input) <= 1:
        return 0,""
    mid = len(input)//2
    left = input[:mid]
    right = input[mid:]

    # print(left,right)
    best_couple=""

    best_left = findbest(left)[0]
    best_right = findbest(right)[0]
    best_cross = max(right) - min(left)
    if max(best_cross,best_right,best_left) == best_cross:
        best_couple = (str(max(right))+'-'+str(min(left)))
        # print(best_couple)


    # if left == input[:len(input)//2]:
    #   print(max(right),'-',min(left))
    # print([max(best_cross,best_right,best_left),max(right),min(left)])
    # print([max(best_cross,best_right,best_left),max(right),min(left)][-1]  )
    # print(max(right),min(left))
    return max(best_cross,best_right,best_left),best_couple



A = [2,5,8,18,12,9,16]
print(findbest(A))
```

时间复杂度分析：
logn 次递归
每次时间复杂度为 n
O(nlogn)

测试结果：

```
(16, '18-2')
[Finished in 0.1s]
```

Counting sort:
解法：

```
def couting_sort(a):
    n=max(a)
    b=[0]*n
    for i in a:
        b[i-1]=b[i-1]+1;

    for i in range(n):
        if b[i] != 0:
            print(i+1)



couting_sort([1,21,32,21,27])
```

时间复杂度分析：
两个并列的 for 循环
一个 for 循化遍历所有元素
一个 for 循环遍历整数范围 k 次
O(n+k)（其中 k 是整数的范围）

测试结果：

```
1
21
27
32
[Finished in 0.1s]
```

Dijkstra 算法：
解法：

```
import heapq
G = {1:{1:0,    2:1,    3:12},
     2:{2:0,    3:9,    4:3},
     3:{3:0,    5:5},
     4:{3:4,    4:0,    5:13,   6:15},
     5:{5:0,    6:4},
     6:{6:0}}

def dijkstra(G,start):      ###dijkstra算法
    INF = 999
    dis = dict((key,INF) for key in G)    # 初始化每个点的距离
    dis[start] = 0
    ###堆优化

    dis_un = {}    #存放未访问的点的距离
    pq = []    #存放排序后的值
    for node,d in dis.items():    #构造最小堆
        entry = [d,node]
        dis_un[node] = entry
        heapq.heappush(pq,entry)

    print("dis_un:",dis_un)
    print(pq)

    while len(pq)>0:
        v_dis,v = heapq.heappop(pq)    #未访问点中距离最小的点和对应的距离
        for node in G[v]:    #与v直接相连的点
            new_dis = dis[v] + G[v][node]
            if new_dis < dis[node]:    #如果与v直接相连的node通过v到src的距离小于dis中对应的node的值,则用小的值替换
                dis[node] = new_dis    #更新所有点的距离
                dis_un[node][0] = new_dis    #更新未访问的点到start的距离 改变dis_un中的entry从而改变pq
                print("_____________________________")
                print("dis_un:",dis_un)
                print(pq)

    return dis
```

时间复杂度分析：
遍历每个点：n
用堆获取最小值：1
堆跟新值：logn
0（nlogn）

测试结果：

```
dis_un: {1: [0, 1], 2: [999, 2], 3: [999, 3], 4: [999, 4], 5: [999, 5], 6: [999, 6]}
[[0, 1], [999, 2], [999, 3], [999, 4], [999, 5], [999, 6]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [999, 3], 4: [999, 4], 5: [999, 5], 6: [999, 6]}
[[1, 2], [999, 4], [999, 3], [999, 6], [999, 5]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [12, 3], 4: [999, 4], 5: [999, 5], 6: [999, 6]}
[[1, 2], [999, 4], [12, 3], [999, 6], [999, 5]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [10, 3], 4: [999, 4], 5: [999, 5], 6: [999, 6]}
[[10, 3], [999, 4], [999, 5], [999, 6]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [10, 3], 4: [4, 4], 5: [999, 5], 6: [999, 6]}
[[10, 3], [4, 4], [999, 5], [999, 6]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [10, 3], 4: [4, 4], 5: [15, 5], 6: [999, 6]}
[[4, 4], [999, 6], [15, 5]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [8, 3], 4: [4, 4], 5: [15, 5], 6: [999, 6]}
[[15, 5], [999, 6]]
_____________________________
dis_un: {1: [0, 1], 2: [1, 2], 3: [8, 3], 4: [4, 4], 5: [15, 5], 6: [19, 6]}
[[15, 5], [19, 6]]
结果 {1: 0, 2: 1, 3: 8, 4: 4, 5: 15, 6: 19}
```