装 订 线

任课教师：

装 订 线

学号：

姓名：

班级：

装 订 线

# 西 安 电 子 科 技 大 学

**考试时间 120 分钟**

# 试　　题

| 题号 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 总分 |
|---|---|---|---|---|---|---|---|---|
| 分数 | 16 | 20 | 12 | 12 | 16 | 16 | 8 | |
| 得分 | | | | | | | | |

1. 考试形式：闭卷☑　开卷☐
2. 考试日期：　　　年　　　月　　　日(答案直接答在试卷上，不要超出装订线)

**1. Answer T/F for the following: (2 * 8 points)**

**(1) $5n2 - 2n + 1024 \in O(n)$** F

**(2) $5n2 - 2n + 1024 \in \Omega(n)$** T

**(3) $5n2 - 2n + 1024 \in \Theta(n2)$** T

**(4) The array A**

**87　85　72　84　79　75　70　55　68**

**forms a max-heap.** F

**(5) If a sorting algorithm is NOT stable, then the output of the algorithm may be NOT in correctly sorted order.** F

**(6) In Dynamic Programming strategy, optimal substructure means that an optimal solution to the problem contains within it an optimal solution to subproblems.** T

**(7) Greedy strategy can be used to obtain an optimal solution of the 0-1 knapsack problem by choosing the most value/ weight per unit in descending order.** F

**(8) 3-COLOR problem is an NP-Complete problem.**

**2．Single Choice (2*10 points)**

**(1) The average-case running time of Insertion Sort is (　　)**

**A. $\Theta(n^2)$**　　**B. $\Theta(nlgn)$**　　**C. $\Theta(n)$**　　**D. $\Theta(n^3)$**

**(2) The worst-case running time of Merge Sort is (　　)**

**A. $\Theta(n^2)$**　　**B. $\Theta(nlgn)$**　　**C. $\Theta(n)$**　　**D. $\Theta(n^3)$**

**(3) The worst-case running time of Quick Sort is (　　)**

**A. $\Theta(n^2)$**　　**B. $\Theta(nlgn)$**　　**C. $\Theta(n)$**　　**D. $\Theta(n^3)$**

**(4) Which of the following sorting algorithm is NOT stable (　　)**

**A. Heap Sort**　　**B. Merge Sort**

C. Insertion Sort D. Counting Sort

(5) Which of the following sorting algorithm is NOT in place ( )

A. Quick Sort B. Bucket Sort

C. Heap Sort D. Insertion Sort

(6) Which designing strategy is used in Quick Sort ( )

A. Divide and conquer B. Dynamic programming

C. Greedy D. Brute Force

(7) Which designing strategy is used in Assembly-Line Scheduling problem ( )

A. Divide and conquer B. Dynamic programming

C. Greedy D. Brute Force

(8) Which designing strategy is used in Activity Selection problem ( )

A. Divide and conquer B. Dynamic programming

C. Greedy D. Brute Force

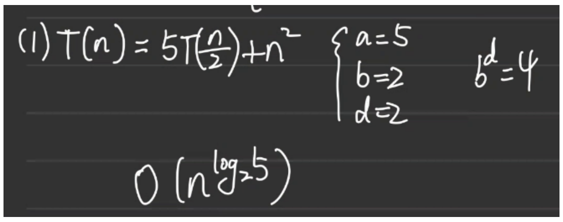
(9) In the DP recursive equation used for Longest Common Subsequence problem, *c[i,j]* represents the ( B ) of *x[1..i]* and *y[1..j]*, it's the ( D ) of the problem.

A. longest common subsequence B. length of longest common subsequence

C. optimal solution D. value of optimal solution

3. Evaluate the following recursions using the Master Method (3 * 4 points)

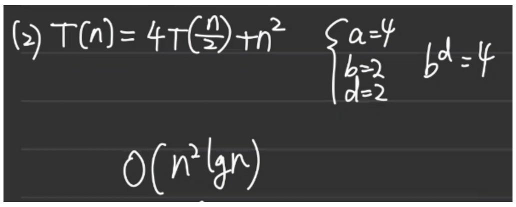
(1) $T(n) = 5T(n/2) + n^2$



$(1)\ T(n) = 5T(\frac{n}{2}) + n^2 \quad \begin{cases} a=5 \\ b=2 \\ d=2 \end{cases} \quad b^d = 4$

$O(n^{\log_2 5})$

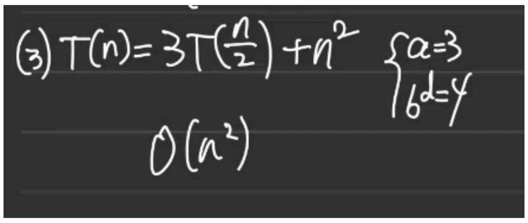
(2) $T(n) = 4T(n/2) + n^2$



$(2)\ T(n) = 4T(\frac{n}{2}) + n^2 \quad \begin{cases} a=4 \\ b=2 \\ d=2 \end{cases} \quad b^d = 4$

$O(n^2 \lg n)$

(3) $T(n) = 3T(n/2) + n^2$



$(3)\ T(n) = 3T(\frac{n}{2}) + n^2 \quad \begin{cases} a=3 \\ b^d=4 \end{cases}$

$O(n^2)$

**4. Divide and Conquer Strategy (12 points)**

**(1) Describe the 3 steps used in Divide and Conquer strategy to solve a problem.**

**(2) Given the following array A to be sorted using Quick Sort as following:**

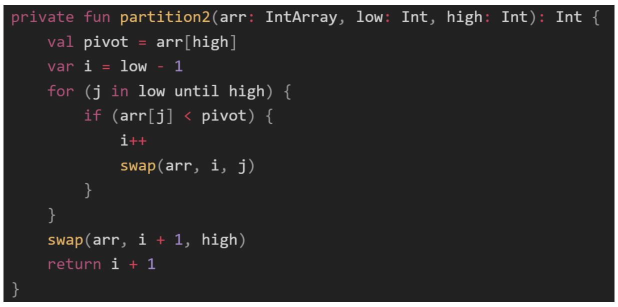
**10 9 8 5 4 11 7 6**

**using the last element (6) as pivot, give the result of the total array after the first partition.**

**(3) Mark the two sub-problems remained in question (2), describe the following steps in Quick Sort for this instance.**

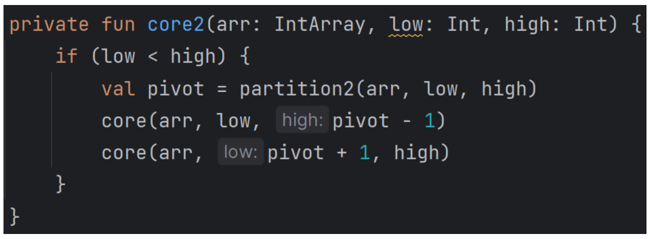
(1) 1. 把大问题分成性质相同，规模更小的子问题；
2. 递归求解子问题；
3. 将子问题的解合并成原来的大问题的解。

(2) 5 4 6 10 9 11 7 8
↑
i



```
private fun partition2(arr: IntArray, low: Int, high: Int): Int {
    val pivot = arr[high]
    var i = low - 1
    for (j in low until high) {
        if (arr[j] < pivot) {
            i++
            swap(arr, i, j)
        }
    }
    swap(arr, i + 1, high)
    return i + 1
}
```

(3) 对5 4和10 9 11 7 8继续做partition操作，然后继续对左右子结构进行操作。直到序列长度为1时不进行任何操作。所有的partition操作都完成时，序列变为有序。



```
private fun core2(arr: IntArray, low: Int, high: Int) {
    if (low < high) {
        val pivot = partition2(arr, low, high)
        core(arr, low, high: pivot - 1)
        core(arr, low: pivot + 1, high)
    }
}
```

**5. Design Strategies(16 points)**

**Answer the following questions briefly.**

**(1) To calculate the optimal solution for Fractional Knapsack problem and 0-1 Knapsack problem, which problem can solved using Greedy strategy? Which problem can be solved using Dynamic Programming strategy? Describe the main idea of the two corresponding algorithms.**

**(2) When solving the Single-Source Shortest Path problem, there may be negative edge(s) or not, in which case Greedy strategy (Dijkstra) can be used to get the optimal solution? Describe the main idea of Dijkstra algorithm. How to calculate the length of the shortest paths in the other case?**

(1) Fractional背包问题使用贪心算法，01背包问题使用动态规划算法。
贪心算法总是在不断寻找局部最优解，并累加到之前的最优解上。这样在问题求解完成时就能得到全局的最优解；动态规划算法将大问题不断划分成小问题，并且大问题的最优解也包括了小问题的最优解。然后，使用Bottom-up的方式进行计算，逐渐得到全局的最优解。贪心算法不能保证求得的最后解是最佳的，而动态规划本质是穷举法，可以保证结果是最佳的。

(2) Dijkstra算法只能解决不存在负权边的单源最短路径问题。思想：从源节点出发，查看它和所有与它相邻的节点的距离，从中找出最短的那一个，并将它标记为已访问。之后，从这个节点开始继续这个过程。注意，每次找最短的路径时，都要从所有还未访问的节点中寻找。当所有结点都标记为已访问时，算法结束。如果图中存在负权边，那么可以使用Bellman-Ford算法来计算。

**6. Maximum Subarray Problem (16 points)**

**Maximum Subarray Problem means to find the subarray indexed from *i* to *j* which maximize the sum of $A_i \sim A_j$ . You should notice that there may be negative ones but not all in the input array A[1..n].**

**(1) Give an algorithm to calculate the maximum subarray in $O(n^2)$ time, using Brute-Force or Divide and Conquer strategy as you like.**

**(2) How to solve the problem in O(n) time using Dynamic Programming strategy? Give and describe the recursive equation used in the optimal substructure of the DP algorithm.**

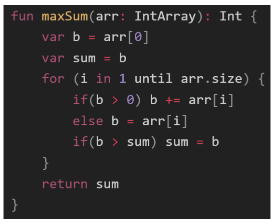
**(3) Given the following input array as following, used the algorithm in step (2) to calculate the maximum subarray.**

**8 -5 -4 10 -1 7 -3 12 -20 18**

(1) Bruce Force：起点从第一个数字迭代到最后一个数字，终点在其中也从第一个数字迭代到最后一个数字。每次都将数字累加到一个临时最优解中，并和全局最优解比较。如果更大，那就会被记录进去。当双重迭代结束之后，全局最优解就是答案。

Divide and Conquer：算出左边，右边和跨越的值，返回三者中的最大值。

(2) $b[j]=max( b[j-1]+a[j], a[j] ), 1\leq j\leq n.$



```
fun maxSum(arr: IntArray): Int {
    var b = arr[0]
    var sum = b
    for (i in 1 until arr.size) {
        if(b > 0) b += arr[i]
        else b = arr[i]
        if(b > sum) sum = b
    }
    return sum
}
```

班级：________ 装 订 线 姓名：________ 学号：________ 装 订 线 任课教师：________ 装 订 线

**7. Algorithm Design(8 points)**

**Design an algorithm to find the longest palindrome(回文) subsequence for a given string S. A subsequence is a sequence that can be derived from a string by deleting some chars without changing the order of the remaining ones. A palindrome is a symmetrical string, that is, a string read identically from left to right as well as from right to left.**

**For example, one longest palindrome subsequence of string "DABBEAF" is "ABBA", whose length is 4.**

**( NOTICE: You're NOT allowed to use the algorithm for Longest Common Subsequence problem! )**

Leetcode 516