数据结构第一次作业

学号：2022212408 姓名：胡宇杭

2023 年 9 月 23 日

1 选择题

1. 从逻辑上可以把数据结构分为（**C**）两大类

   a. 动态结构、静态结构

   b. 顺序结构、链式结构

   c. 线性结构、非线性结构

   d. 初等结构、构造型结构

2. 下面关于算法的说法正确的是（**D**）

   a. 算法的时间复杂度一般与算法的空间复杂度成正比

   b. 解决某问题的算法可能有多种，但肯定采用相同的数据结构

   c. 算法的可行性是指算法的指令不能有二义性

   d. 同一个算法，一般情况下实现语言的级别越高，执行效率越低。

3. 在发生非法操作时，算法能够做出适当处理的特性称为（**B**）

   a. 正确性

   b. 健壮性

   c. 可读性

d. 可移植性

4. 在存储数据时，通常不仅要存储各数据元素的值，而且还要存储（ **C** ）

   a. 数据的处理方法

   b. 数据元素的类型

   c. 数据元素之间的关系

   d. 数据的存储方法

5. 给定 $N \times N$ 的二维数组 A，则在不改变数组的前提下，查找数组中最大元素的时间复杂度是（ **A** ）

   a. $O(N^2)$

   b. $O(NlogN)$

   c. $O(N)$

   d. $O(N^2logN)$

2 判断题

1. 数据的逻辑结构是指数据的各数据项之间的逻辑关系（ X ）
2. 顺序存储方式的优点是存储密度大，且插入、删除运算效率高（ X ）
3. 数据的逻辑结构说明数据元素之间的次序关系，它依赖于数据的存储结构（ X ）
4. 算法的优劣与描述算法的语言无关，但与所用的计算机性能有关（ X ）
5. 算法必须有输出，但可以没有输入（ $\surd$ ）
6. 若 $f(n) = O(n^2)$，$g(n) = O(n)$ 那么 $f(n) + g(n) = O(n^2)$（ $\surd$ ）
7. 若 $f(n) = O(n^2)$，$g(n) = O(n)$ $f(n) * g(n) = O(n^3)$（ $\surd$ ）

3 简答题

1. 分析程序段中带“#”语句的执行频度，并给出程序段的时间复杂度

   a. 第一小题

```
j = 1; k = 0;
while (j <= n - 1){
    j ++ ;
    k += j; // #
}
```

“#”语句在 *while* 循环的每一次迭代中都会执行一次，因此，*while* 循环的迭代次数即为该语句的执行频度。下面分析 *while* 的迭代次数：

while 循环的结束条件为 $j > n-1$，而每一次迭代时 j 的值都会增加 1，直至达到结束条件。j 的初始值是 1，因此 *while* 循环共迭代 $n-1$ 次，因此该语句的执行频度为 $n-1$。

算法共执行了 $n-1$ 次 *while* 循环，每次循环执行两条语句，所以时间复杂度为 $O(2(n-1)) = O(n)$。

   b. 第二小题

```
// 设 n 是偶数
for (i = 1, s = 0; i <= n; i ++ ){
    for (j = 2 * i; j <= n; j ++ ){
        s ++ ; // #
    }
}
```

当 $i \leq \frac{n}{2}$ 时，内层循环每次会执行 $n-2i+1$ 次，当 $i > \frac{n}{2}$ 时，j 的初始值为 $2i > n$，此时不会进行内层循环，故内层循环共进行了 $\sum_{i=1}^{\frac{n}{2}}(n-2i+1) = \frac{n(n-1)}{2}$ 次，即为该语句的执行频度。

算法共执行了 $\frac{n^2}{4}$ 次，时间复杂度为 $O(\frac{n^2}{4}) = O(n^2)$。

   c. 第三小题

```
k = 0;
for (i = 0; i < n; i ++ ){
```

```
3       for (j = i; j < n; j ++ ){
4           k ++ ; // #
5       }
6   }
```

外层循环中，i 从 0 开始递增，共迭代了 n 次，内层循环中 j 从 i 开始递增，共迭代了 $n-i$ 次，故该语句的执行频度为 $\sum_{i=0}^{n-1} n-i = \frac{n(n+1)}{2}$，时间复杂度为 $O(\frac{n(n+1)}{2}) = O(n^2)$。

d. 第四小题

```
1   k = 0;
2   for (i = 0; i < n; i ++ ){
3       for (j = 1; j < n; j <<= 1){
4           k ++ // #
5       }
6   }
```

外层循环中，i 从 0 开始递增，共迭代了 n 次，内层循环中 j 从 1 开始，每次迭代左移一位，即 $j* = 2$，共迭代了 $\lceil log_2 n \rceil$ 次，故语句的执行频度为 $n\lceil log_2 n \rceil$，时间复杂度为 $O(n\lceil log_2 n \rceil) = O(nlog_2 n)$。

2. 有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。假定对包含 $1, 2, 3, 4, 5$ 共五个元素的序列 (即 $n = 5$) 做冒泡排序，请举例说明何时会出现以下情况：

   a. 任何元素都无需移动

   当序列为 **1, 2, 3, 4, 5** 时无需移动。因为对于每一对相邻元素，其顺序都是正确的，无需交换。

   b. 某元素会一度朝着远离预期最终位置的方向移动

   当序列为 **5, 4, 2, 3, 1** 时，第一次遍历时会交换 $(5, 4)$，导致 4 向远离预期最终位置的方向移动。

   c. 某元素初始时已位于最终位置，却需要参与 $n-1$ 次交换

   当序列为 **5, 4, 3, 2, 1** 时，所有元素都需要参与 $n-1$ 次交换，而 3 刚好处于最终位置上。

d. 所有元素都需要参与 $n-1$ 次交换

序列 **5, 4, 3, 2, 1**

3. 阅读高质量 $C++/C$ 编程指南或其他 C 编程资料，回答以下问题

a. 头文件的作用是什么，头文件中为什么有 $ifndef/define/endif$ 结构的预处理块?

头文件的作用：

- **声明和定义：**头文件通常包含函数声明、变量声明、类型定义（如结构和类）以及模板定义。这允许多个源文件共享相同的声明和定义，从而实现代码的模块化和重用；
- **代码组织：**头文件提供了一种将代码组织成逻辑单元的方法，使得代码更易于管理和维护；
- **参数化编码：**通过宏定义和条件编译，头文件可以用于参数化编码，从而为不同的目标和平台生成不同的代码版本。

预处理块的作用：其目的是防止头文件被多次包含，从而导致重复的声明或定义，进而导致编译错误。

b. 引用和指针有何区别?下面代码中的 $Test$ 函数的语句 $GetMemory(str, 200)$ 并没有使 str 获得期望的内存，str 依旧是 $NULL$，请问是为什么，应该如何修改?

```
1  void GetMemory(char *p, int num)
2  {
3      p = (char *)malloc(sizeof(char) * num);
4  }
5
6  void Test(void)
7  {
8      char *str = NULL;
9      GetMemory(str, 100);  // str 仍然为 NULL
10     strcpy(str, "hello"); // 运行错误
11 }
```

本质上来说，引用是一个 **常量指针**；从定义上来看，指针是一个变量，其值为另一个变量的地址，其空值可以是 $NULL$，可以在其生命周期

中指向不同的对象；引用是另一个已存在变量的别名，一旦初始化，引用就不能再指向其他变量，空值不能为 $NULL$，一旦初始化后，就不能重新绑定到另一个对象。

原因：p 是临时变量，在程序中，只是将 str 的值 $(NULL)$ 传给了 p，然后为临时变量 p 开辟了内存空间，并不对 str 产生影响。

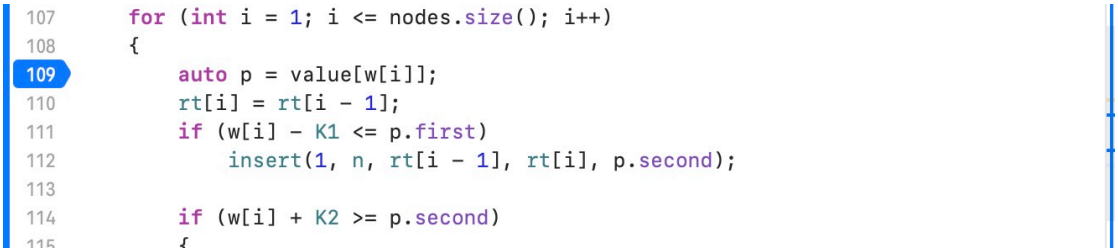
修改：将 $GetMemory()$ 函数中的 $char\ *p$ 改为 $char**p$。κ

不会，函数体内的局部变量会存放在栈中，在函数结束时会自动释放内存；而动态申请的内存为堆区内存，需要手动释放。

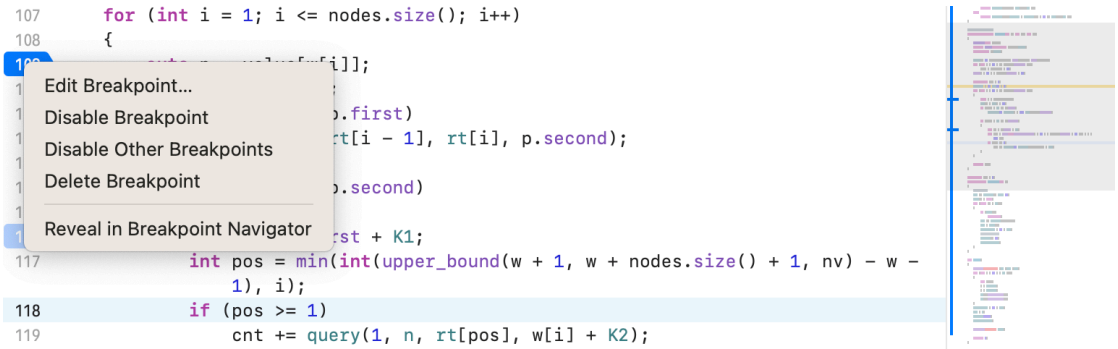
4. 在调试程序时如果循环在执行 100 次后会出现问题，应如何设置断点进行调试？在调试程序时如果需要在被监控变量的值发生改变时停止执行进行分析，应如何设置？

   以 $Xcode$ 为例进行操作：

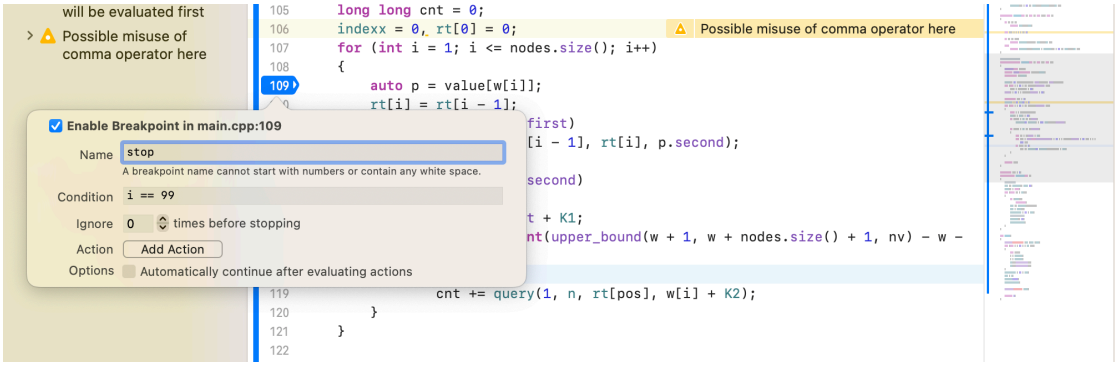
   (a) 先左键点击需要添加断点的位置；



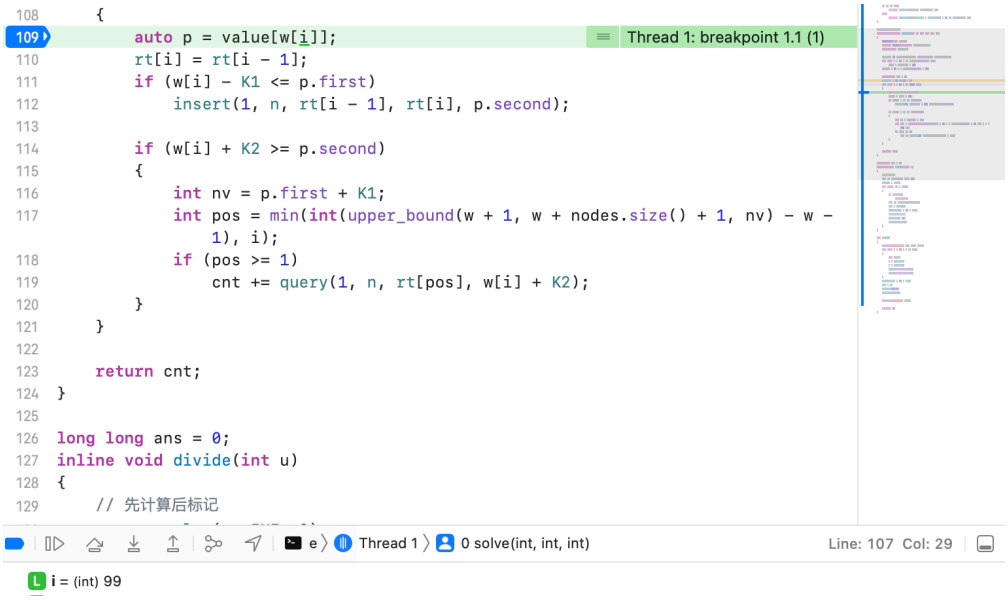
   (b) 右键断点，选择 **Edit Breakpoint...**；



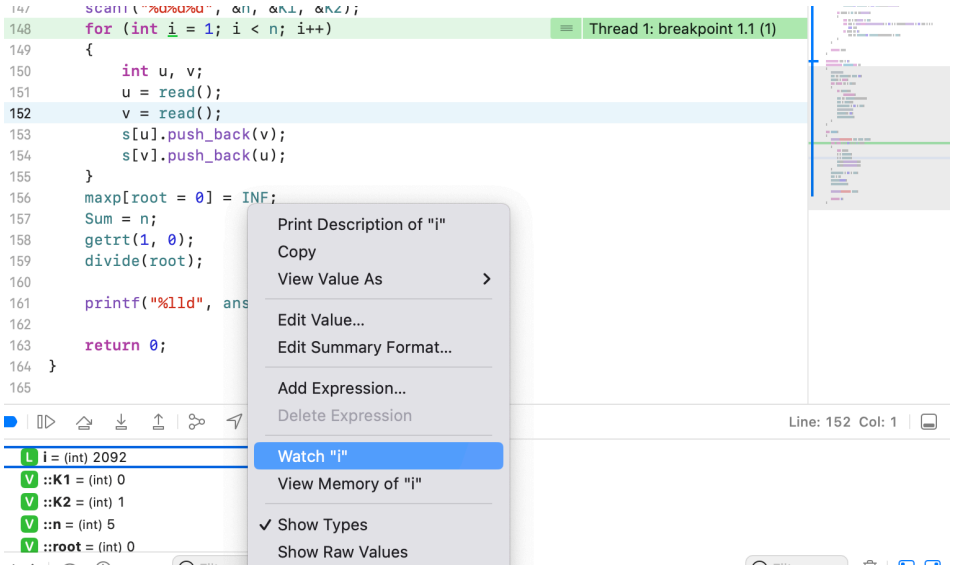
(c) 因为在 $i >= 100$ 时会出现问题，所以在 **Condition** 中输入 **i == 99**;



(d) 继续运行程序，发现 **i == 99** 时程序停止。



(e) 接下来设置监视，在程序运行时在监视窗口左键点击想要监视的变量；右键，选择 **Watch “elem”**；



(f) 继续运行，发现在 **i** 发生变化时程序暂停，并且输出了 **i** 的变化值。