

**C++** 五级

2024 年 09 ⽉

**1** 单选题 (每题 **2** 分,共 **30** 分)

**第 1 题** 下⾯关于链表和数组的描述 ，错误的是（ ）。

口 **A.** 数组⼤⼩固定 ，链表⼤⼩可动态调整。

口 **B.** 数组⽀持随机访问 ，链表只能顺序访问。

口 **C.** 存储相同数⽬的整数 ，数组⽐链表所需的内存多。

口 **D.** 数组插⼊和删除元素效率低 ，链表插⼊和删除元素效率⾼ 。

**第 2 题** 通过（ ）操作 ，能完成在双向循环链表结点 p 之后插⼊结点 s 的功能（其中 next 域为结点的直接后继， prev 域为结点的直接前驱） 。

 **A.** p->next->prev = s; s->prev = p; p->next = s; s->next = p->next;

口 **B.** p->next->prev = s; p->next = s; s->prev = p; s->next = p->next;

口 **C.** s->prev = p; s->next = p->next; p->next = s; p->next->prev = s;

 **D.** s->next = p->next; p->next->prev = s; s->prev = p; p->next = s; **第 3 题** 对下⾯两个函数 ，说法错误的是（ ）。

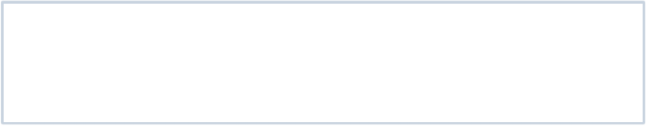
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int sumA(int n) {  int res = 0;  for (int i = 1; i <= n; i++) { res += i;  }  return res; }  int sumB(int n) {  if (n == 1)  return 1;  int res = n + sumB(n - 1);  return res; } |

口 **A.** sumA体现了迭代的思想。 口 **B.** SumB采⽤的是递归⽅式。

口 **C.** SumB函数⽐SumA的时间效率更⾼ 。

口 **D.** 两个函数的实现的功能相同。

**第 4 题** 有如下函数 fun ，则 fun(20, 12) 的返回值为（ ）。

1

2

3

4

5

6

int fun(int a, int b) {

if (a % b == 0)

return b; else

return fun(b, a % b); }

口 **A.** 20

口 **B.** 12

口 **C.** 4

口 **D.** 2

**第 5 题** 下述代码实现素数表的埃拉托斯特尼筛法 ，筛选出所有⼩于等于 n 的素数 ，则横线上应填的最佳代码是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | void sieve\_Eratosthenes(int n) {  vector<bool> is\_prime(n + 1, true);  vector<int> primes;  for (int i = 2; i \* i <= n; i++) {  if (is\_prime[i]) {  primes.push\_back(i);  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ { // 在此处填入代码 is\_prime[j] = false;  }  }  }  for (int i = sqrt(n) + 1; i <= n; i++) {  if (is\_prime[i]) {  primes.push\_back(i);  }  }  return primes; } |

口 **A.** for (int j = i; j <= n; j++)

口 **B.** for (int j = i \* i; j <= n; j++)

 **C.** for (int j = i \* i; j <= n; j += i)

 **D.** for (int j = i; j <= n; j += i)

**第 6 题** 下述代码实现素数表的线性筛法 ，筛选出所有⼩于等于 n 的素数 ，则横线上应填的代码是( )。



vector<int> sieve\_linear(int n) {

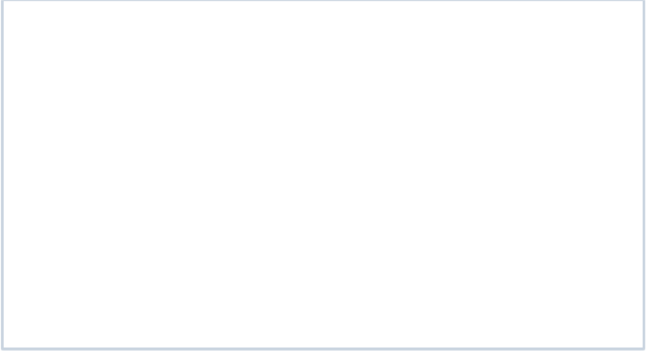
vector<bool> is\_prime(n + 1, true);



1

2



3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

vector<int> primes;

for (int i = 2; i <= n / 2; i++) {

if (is\_prime[i])

primes.push\_back(i);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ { // 在此处填入代码 is\_prime[i \* primes[j]] = 0;

if (i % primes[j] == 0)

break;

}

}

for (int i = n / 2 + 1; i <= n; i++) {

if (is\_prime[i])

primes.push\_back(i); }

return primes; }

 **A.** for (int j = 0; j < primes.size() && i \* primes[j] <= n; j++) 口 **B.** for (int j = 1; j < primes.size() && i \* j <= n; j++)

 **C.** for (int j = 2; j < primes.size() && i \* primes[j] <= n; j++) 口 **D.** 以上都不对

**第 7 题** 下⾯函数可以将 n 的所有质因数找出来 ，其时间复杂度是（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <iostream> #include <vector>  vector<int> get\_prime\_factors(int n) { vector<int> factors;  while (n % 2 == 0) {  factors.push\_back(2);  n /= 2; }  for (int i = 3; i \* i <= n; i += 2) {  while (n % i == 0) {  factors.push\_back(i); n /= i;  }  }  if (n > 2) {  factors.push\_back(n); }  return factors; } |

口 **A.** O(n2)

口 **B.** O(nlogn)

口 **C.** n log n)

口 **D.** O(n)

**第 8 题** 现在⽤如下代码来计算cn（n个c相乘） ，其时间复杂度为（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5 | double quick\_power(double x, unsigned n) { if (n == 0) return 1;  if (n == 1) return x;  return quick\_power(x, n / 2) \* quick\_power(x, n / 2) \* ((n & 1) ? x : 1); } |

口 **A.** O(n)

**B.** o( 2

n

口 **C.** log n)

口 **D.** O(nlogn)

**第 9 题** 假设快速排序算法的输⼊是一个长度为n的已排序数组 ，且该快速排序算法在分治过程总是选择第一个元素 作为基准元素 。下⾯选项（ ）描述的是在这种情况下的快速排序⾏为。

口 **A.** 快速排序对于此类输⼊的表现最好， 因为数组已经排序。

口 **B.** 快速排序对于此类输⼊的时间复杂度是o(nlogn) 。

口 **C.** 快速排序对于此类输⼊的时间复杂度是O(n2) 。

 **D.** 快速排序⽆法对此类数组进⾏排序， 因为数组已经排序。

**第 10 题** 考虑以下C++代码实现的归并排序算法：

void merge(int arr[], int left, int mid, int right) { int n1 = mid - left + 1;

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

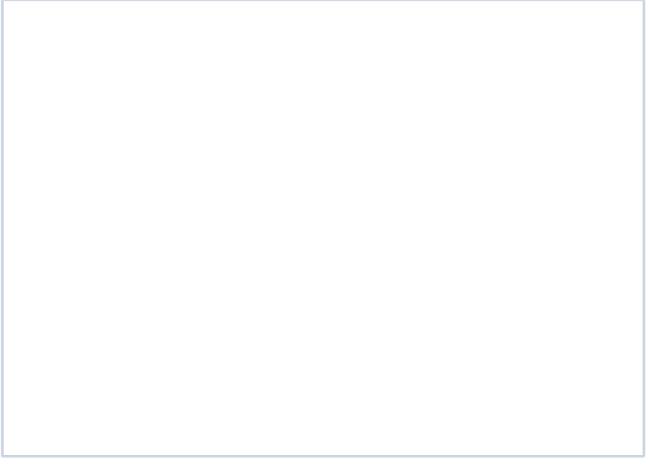
else {

arr[k] = R[j];

j++;

}

k++;

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void merge\_sort(int arr[], int left, int right) { if (left < right) {

int mid = left + (right - left) / 2;

merge\_sort(arr, left, mid);

merge\_sort(arr, mid + 1, right);

merge(arr, left, mid, right); }

}

对长度为 n 的数组 arr ，挑⽤函数 merge\_sort(a, 0, n-1) ，在排序过程中 merge 函数的递归调⽤次数⼤约是 ( ) 。

口 **A.** o(1)

口 **B.** O(n)

口 **C.** log n)

口 **D.** O(nlogn)

**第 11 题** 现在有 n 个⼈要过河 ，每只船最多载2⼈ ，船的承重为100kg 。下列代码中 ，数组 weight 中保存有 n 个⼈ 的体重（单位为kg）， 已经按从⼩到⼤排好序 ，代码输出过河所需要的船的数⽬ ，采⽤的思想为（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int i, j;  int count = 0;  for (i = 0, j = n - 1; i < j; j--) {  if (weight[i] + weight[j] <= 100) {  i++; }  count++;  }  printf("过河的船数 ：%d\n", count); |

口 **A.** 枚举算法 口 **B.** 贪⼼算法

口 **C.** 迭代算法

口 **D.** 递归算法

**第 12 题** 关于分治算法， 以下哪个说法正确？

口 **A.** 分治算法将问题分成⼦问题 ，然后分别解决⼦问题 ，最后合并结果。

口 **B.** 归并排序不是分治算法的应⽤ 。

口 **C.** 分治算法通常⽤于解决⼩规模问题。

口 **D.** 分治算法的时间复杂度总是优于O(nlog(n)) 。

**第 13 题** 根据下述⼆分查找法 ，在排好序的数组 1， 3，6，9，17， 31， 39， 52，61，79 中查找数值 31 ，循环 while (left <= right) 执⾏的次数为（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | int binary\_search(vector<int>& nums, int target) { int left = 0;  int right = nums.size() - 1;  while (left <= right) {  int mid = left + (right - left) / 2;  if (nums[mid] == target) {  return mid;  }  else if (nums[mid] < target) {  left = mid + 1;  }  else {  right = mid - 1;  }  }  return -1; // 如果找不到目标元素 ，返回-1  } |

口 **A.** 1

口 **B.** 2

口 **C.** 3

口 **D.** 4

**第 14 题** 以下关于⾼精度运算的说法错误的是( )。

口 **A.** ⾼精度计算主要是⽤来处理⼤整数或需要保留多位⼩数的运算。

口 **B.** ⼤整数除以⼩整数的处理的步骤可以是 ，将被除数和除数对齐 ，从左到右逐位尝试将除数乘以某个数 ，通过 减法得到新的被除数 ，并累加商。

 **C.** ⾼精度乘法的运算时间只与参与运算的两个整数中长度较长者的位数有关。

口 **D.** ⾼精度加法运算的关键在于逐位相加并处理进位。

**第 15 题** 当n = 7时 ，下⾯函数的返回值为（ ）。

1

2

3

4

5

int fun(int n) {

if (n == 1) return 1;

else if (n >= 5) return n \* fun(n - 2); else return n \* fun(n - 1);

}

口 **A.** 105 口 **B.** 840 口 **C.** 210 口 **D.** 420

**2** 判断题 (每题 **2** 分,共 **20** 分)

**第 1 题** 在操作系统中 ，需要对一组进程进⾏循环 。每个进程被赋予一个时间⽚ ，当时间⽚⽤完时 ，CPU将切换到下 一个进程 。这种循环操作可以通过环形链表来实现。

**第 2 题** 找出⾃然数 n 以内的所有质数 ，常⽤算法有埃拉托斯特尼（埃⽒） 筛法和线性筛法 ，其中线性筛法效率更

⾼ 。

**第 3 题** 唯一分解定理表明任何一个⼤于 1 的整数都可以唯一地分解为素数之和。

**第 4 题** 贪⼼算法通过每一步选择局部最优解 ，从⽽一定能获得最优解。

**第 5 题** 快速排序和归并排序的平均时间复杂度均为(nlog n) ，且都是稳定排序。

**第 6 题** 插⼊排序的时间复杂度总是⽐快速排序低。

**第 7 题** 引⼊分治策略往往可以提升算法效率 。一⽅⾯ ，分治策略减少了操作数量 ；另一⽅⾯ ，分治后有利于系统的

并⾏优化。

**第 8 题** ⼆分查找要求被搜索的序列是有序的 ，否则⽆法保证正确性。

**第 9 题** 在C++语⾔中 ，递归的实现⽅式通常会占⽤更多的栈空间 ，可能导致栈溢出。

**第 10 题** 对于已经定义好的标准数学函数 sin(x) ，应⽤程序中的语句 y=sin(sin(x)); 是一种递归调⽤ 。

**3** 编程题 (每题 **25** 分,共 **50** 分)

