به نام خدا

|  |  |  |  |
| --- | --- | --- | --- |
| مدرس: مظفر بگ محمدي | دانشگاه ايلام | ترم اول سال تحصيلي 98-97 | سری دوم تمرینات ساختمان داده |

1. با استفاده از تئوری Master پیچیدگی محاسباتی (زمانی) حالات زیر را تعیین کنید:
   1. **T(n) = T(n/3) + 2n**
   2. **T(n) = 4T(n/2) + n**
   3. **T(n) = T(n/3) + n**
   4. **T(n) = 4T(n/2) + n3**
   5. **T(n) = 2T(3n/2) + n**
   6. **T(n) = 2T(n/2) +3n2+n**
2. پیچیدگی محاسباتی تابع زیر را به هر دو روش ارائه شده در کتاب (روش شمارنده و روش تشکیل جدول تکرار) تعیین کنید:

void transpose(Matrix a; int n)

{

for (int j = 0; j < (n-1); j++)

{

for (int k = j+1; k < n; k++)

{

MatrixElement t = a[j][k];

a[j][k] = a[k][j];

a[k][j] = t;

}

}

}

1. پیچیدگی محاسباتی قطعه کدهای زیر را محاسبه کنید:

a. if (n > 1) {

return n;

} else {

return 0;

}

b. public int method2(int n) {

int sum = 0;

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

sum += j \* k;

}

}

return sum;

}

c. public int method3(int n) {

int sum = 0;

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

for (int l = 0; l < n; l++) {

sum += j \* k / (l + 1);

}

}

}

return sum;

}

d. public int method4(Integer[] data, int key) {

int startIndex = 0;

int endIndex = data.length - 1;

while (startIndex < endIndex) {

int midIndex = (endIndex - startIndex )/ 2 + startIndex;

int midValue = data[midIndex];

if (key > midValue) {

startIndex = midIndex++;

} else if (key < midValue) {

endIndex = midIndex - 1;

} else {

return midIndex;

}

}

return -1;

}

e. public static boolean method5(String[] args) {

for (int i = 0; i < args.length; i++) {

for (int j = 0; j < args.length; j++) {

if (i == j)

break;

if (args[i] == args[j])

return true;

}

}

return false;

}

f. for(int i=1; i<n; i++)  
       for(int j=0; j < i\*i; j++)  
          if ( (j%2) == 0)  
             a=a+2;

g. float sum=0;  
    for( i=1; i <= n/2; i++)                    
       for ( j=1; j<i\*i; j++)                   
          if (j  %  i == 0)    
                for (k=0; k<n; k++){  
                     sum = sum+1;

     sum\*=n\*(sum+1);

}

h. for (x=0, i=1; i<n; i\*=2)  
       for(int j=1; j <= 2\*n; j++)  
          a++;

1. بازگشت: برنامه زیر را طوری تکمیل کنید که بتواند بزرگترین عنصر یک آرایه را پیدا کند. سپس با استفاده از تئوری master پیچیدگی زمانی برنامه را حساب کنید.

int maxValue(int[] A,int left,int right)

{ \\find max value of A[left]...A[right]

if (left==right)

return ? ;

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

int ans1 = maxValue(A, left, mid);

int ans2 = ?;

?

?

?

?

}

1. فرض کنيد که الگوريتمهاي T و C براي حل يک مساله مشابه با هم رقابت مي کنند. پيچيدگي زماني T از رابطه T(n)=7T(n/2)+n2 پيروي مي کند و پيچيدگي زماني C از رابطه T(n)=aT(n/4)+n2 پيروي مي کند. بزرگترين مقدار a را که به ازاي آن C سريعتر است را بدست آوريد.
2. در این سوال شما یک الگوریتم بازگشتی را تحلیل و پیچیدگی آنرا حساب می کنید. لطفا وقت خود را به دنبال فهمیدن اینکه کد چکار می کند تلف نکنید. چون اهمیتی ندارد! فقط به سوالاتی که در انتهای کد مطرح شده جواب دهید.

int foo(int[] A){

n = A.length;

if (n==1)

return A[0];

/\*\*\* divide \*\*\*/

int half = (int) n/2

int[] A1 = new int[half];

int[] A2 = new int[n-half];

int[] A3 = new int[half]

for (int i=0; i < half; i++){

for (int j=0; j <= 1; j++){

if (j==0) A1[i] = A[2i];

else A2[i] = A[2i+1];

}

}

A2[n-half-1] = A[n-1];

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

A3[i] = A1[i]\*A2[i];

/\*\*\* end of divide \*\*\*/

b1 = foo(A1);

b2 = foo(A2);

b3 = foo(A3);

if (b3 > b1\*b2)

return b3;

else

return b1\*b2;

}

* 1. پیچیدگی زمانی تقسیم مساله اصلی به زیر مسائل آن چقدر است؟
  2. یک رابطه بازگشتی بنویسید که ارتباط پیچیدگی زمانی foo را (T(n)) با زیر مسائل آن بیان می کند؟
  3. با استفاده از تئوری master ، مقدار T(n) را بدست آورید.

1. زمان اجرا: ثابت کنید که پیچدگی زمانی T(n) = 2n3 + n2 + log n + 10100 از درجه ی O(n3) است.