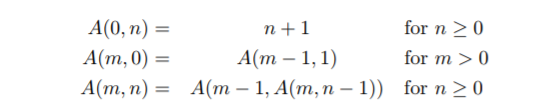
به نام خدا

|  |  |  |  |
| --- | --- | --- | --- |
| مدرس: مظفر بگ محمدي | دانشگاه ايلام | ترم اول سال تحصيلي 01-00 | سری اول تمرینات ساختمان داده |

1. یک برنامه ی بازگشتی بنویسید که اعداد x و n را بگیرد و xn را محاسبه کند.
2. یک برنامه ی بازگشتی بنویسید که تعداد راههای خورد کردن یک اسکناس هزارتومانی را به سکه های 500، 100 و 50 تومانی محاسبه کند.
3. یک برنامه ی بازگشتی بنویسید که بزرگترین مقسوم علیه مشترک را به روش اقلیدس (نردبانی) محاسبه کند.
4. یک برنامه ی بازگشتی بنویسید که تابع آکرمان را محاسبه کند. تعریف این تابع به صورت زیر است:



1. یک برنامه بازگشتی بنویسید یک آرایه از اعداد طبیعی دریافت کند و به صورت زیر عمل کند :

Input: A = {1, 2, 3, 4, 5}

Output: [48]

[20, 28]

[8, 12, 16]

[3, 5, 7, 9]

[1, 2, 3, 4, 5]

راهنمایی: در اولین چرخش تمام عناصر هستند و سپس هر عنصر با عنصر بعدی خود جمع میشود

Guidance :

[48]

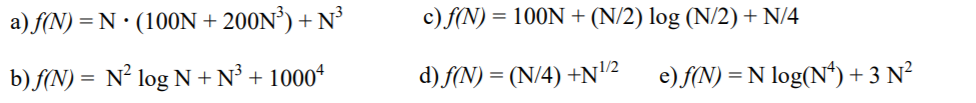
[20, 28] 🡪(20 + 28 = 48)

[8, 12, 16] 🡪 (8 + 12 = 20, 12 + 16 = 28)

[3, 5, 7, 9] 🡪 (3 + 5 = 8, 5 + 7 = 12, 7 + 9 = 16)

[1, 2, 3, 4, 5] 🡪 (1 + 2 = 3, 2 + 3 = 5, 3 + 4 = 7, 4 + 5 = 9)

1. نشان دهید اگر f1(n)=Ω(g1(n)) و f2(n)=Ω(g2(n)) باشد، آنگاه f1(n)×f2(n)= Ω(g1(n)×g2(n)) است.
2. برای هر یک از عبارات زیر کمترین O ممکن را بدست آورید:

****

1. با استفاده از تئوری Master پیچیدگی محاسباتی (زمانی) حالات زیر را تعیین کنید:
   1. **T(n) = 3T(n/3) + n3**
   2. **T(n) = 3T(n/2) + n**
   3. **T(n) = 2T(n/3) +3n2+n**
   4. **T(n) = T() + 1**
   5. **T(n) = 8T() +**
   6. **T(n) = T() +**
2. ثابت کنید که پیچدگی زمانی T(n) = 12n3 + 2n2 + log n از درجه ی O(n3) است.
3. پیچیدگی محاسباتی قطعه کدهای زیر را محاسبه کنید:
   1. int count = 0;

for (int i = N; i > 0; i /= 2)

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

count++;

* 1. void function(int n) {

int count = 0;

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

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

if (j%i == 0)

{

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

printf("\*");

}

}

* 1. function(int n) {

if (n==1)

return;

for (int i=1; i<=n; i++)

{

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

{

printf("\*");

break;

}

}

}

* 1. void silly(int n) {

if (n <= 0) return;

System.out.println(”n = ” + n);

silly(n/2);

}

* 1. void fun() {

int i, j;

for (i=1; i<=n; i++)

for (j=1; j<=log(i); j++)

printf("\*");

}

* 1. void fun() {

int i, j;

for (i=1; i<=n; i -= )

printf("\*");

}

* 1. void ds981() {

int count = 0;

for (int i=1; i<=n; i++)

for (int j=i; j<=n; j++)

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

count++;

}

* 1. void ds981() {

int count = 0;

for (int i=1; i<=n; i++)

for (int j=i; j<=n; j = 2 \* j)

{

k = j;

r = 1;

while (r < k)

{

Count++;

r++;

}

}

}

1. 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.