Домашнее задание 2
по алгоритмам LaTeX

Попов Николай

19 февраля 2018 г.

№1

$$\sum_{i=1}^{n} \sqrt{i^3+2i+5} \leqslant n\sqrt{n^3+2n+5} \leqslant n\sqrt{4n^3} = 2n^{\frac{5}{2}}$$

$$\sum_{i=1}^{n} \sqrt{i^3+2i+5} \geqslant \sum_{i=n/2}^{n} \sqrt{i^3+2i+5} \geqslant \frac{n}{2}\sqrt{\left(\frac{n}{2}\right)^3+n+5} \geqslant \frac{n}{2}\sqrt{\left(\frac{n}{2}\right)^3} = \frac{1}{4\sqrt{2}}n^{\frac{5}{2}}$$

Отсюда заключаем, что

$$\sum_{i=1}^{n} \sqrt{i^3+2i+5} = \Theta(n^{\frac{5}{2}})$$

№2

Ответ: да.
(в предположении что функции класса o(1) - положительные)

$$f(n) = (3+o(1))^n + \Theta(n^{100}) \geqslant 3^n + \Theta(n^{100}) \geqslant 3^n \Rightarrow \log f(n) \geqslant n \log 3$$

Расммотрим произвольное $C \in R$, C $>$0. Тогда

$$f(n) = (3+o(1))^n + \Theta(n^{100}) \leqslant (3+C)^n + \Theta(n^{100}) \leqslant 2^n(3+C)^n = (6+2C)^n$$

$$\Rightarrow \log f(n) \leqslant n \log{(6+2C)}$$

Т.к. $\Theta(n^{100}) \leqslant (3+C)^n \Rightarrow \Theta(n^{100}) \leqslant (2^n-1)(3+C)^n$ В итоге, $\log f(n) = \Theta(n)$

№3

Пусть $k^2 < n < (k+1)^2$

$$g(n) = \sum_{b=1}^{k}\left[b\log n + \sum_{i=0}^{b-1}\frac{i}{2}\right] = \frac{1}{2}k(k+1)logn + \frac{1}{4}\sum_{b=1}^{k} b(b-1) =$$

$$= \frac{1}{2}k(k+1)logn + \frac{1}{4}\left(\frac{1}{6}k(k+1)(2k+1) - 1/2k(k+1)\right) =$$

$$= \frac{1}{2}k(k+1)\left(logn + \frac{1}{12}(2k+1) - \frac{1}{4}\right)$$

$$k^2 < n < (k+1)^2 \Leftrightarrow \frac{\sqrt{n}}{2} \leqslant \sqrt{n} - 1 < k < \sqrt{n} \Leftrightarrow \frac{n}{4} \leqslant k^2 \leqslant n$$

Тогда $k = \Theta(\sqrt{n})$ и $k^2 = \Theta(n)$. А значит,

$$\frac{1}{2}k(k+1) = \Theta(n)$$

$$\left(logn + \frac{1}{12}(2k+1) - \frac{1}{4}\right) = \Theta(\sqrt{n})$$

В итоге, $g(n) = \Theta(n\sqrt{n}) = \Theta(n^{3/2})$

№4

а) $238x + 385y = 133 \Leftrightarrow 34 * 7x + 55 * 7y = 19 * 7 \Leftrightarrow 34x + 55y = 19$

Решим

$$34x + 55y = 1$$

$$34\begin{pmatrix}1\\0\end{pmatrix} 55\begin{pmatrix}0\\1\end{pmatrix}$$

$$34\begin{pmatrix}1\\0\end{pmatrix} 21\begin{pmatrix}-1\\1\end{pmatrix}$$

$$13\begin{pmatrix}2\\-1\end{pmatrix} 21\begin{pmatrix}-1\\1\end{pmatrix}$$

$$13\begin{pmatrix}2\\-1\end{pmatrix} 8\begin{pmatrix}-3\\2\end{pmatrix}$$

$$5\begin{pmatrix}5\\-3\end{pmatrix} 8\begin{pmatrix}-3\\2\end{pmatrix}$$

$$5\begin{pmatrix}5\\-3\end{pmatrix} 3\begin{pmatrix}-8\\5\end{pmatrix}$$

$$2\begin{pmatrix}13\\-8\end{pmatrix} 3\begin{pmatrix}-8\\5\end{pmatrix}$$

$$2\begin{pmatrix}13\\-8\end{pmatrix} 1\begin{pmatrix}-21\\13\end{pmatrix}$$

Получили решение (-21 13).
Тогда уравнение $34x+55y = 19$ имеет решение (-21*19 13*19)= (-399 247)
А уравнение $238x + 385y = 133$ имеет общее решение

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -399 \\ 247 \end{pmatrix} + C \begin{pmatrix} 55 \\ -34 \end{pmatrix}$$

где C - целое число. б)Уравнение $143x + 121y = 52$ не имеет решений в целых числах, т.к. 11=(143,121) не является делителем 52.

№5

1)Корректность: Т.к. указанный алгоритм возврашает правильный результат при x=y=0, то осталось убедить в правильности рекурентного перехода:
Рассмотрим 2 ситуации i-того шага рекурсии - когда x на этом шаге нечетный и четный. Предположим, что Divide($[\frac{x}{2}], y$) вернул правильные q' и r', такие что $x' = q'y + r'$. Зная, что $x = 2x'$, запишем $x = 2q'y + 2r'$. При этом, $q = 2q'$ и $r = 2r'$. $r' < y => r = 2r' < 2y => r_2 = r - y < y$ - т.е. если r>y , число r-y будет остатком от деления. В этом случае также надо увеличить частное q. Аналогично поступаем при нечетном x. Получим $x = 2x' + 1 = 2q'y + 2r' + 1$. Если r = 2r'-1 > y, то вычтем из него y. Тогда $r' < y => r = 2r' + 1 < 2y + 1 => r - y < y + 1 => r - y < y$, т.к. если $r - y = y$, то $r = 2r' + 1 = 2y$, что невозможно в целых числах. Т.к. алгоритм содержит именно такую последовательность действий, то он правильный.
2)Рекурсивных вызовов будет порядка $\log_2 x = n$(длина двоичного слова x), в каждом из которых производится умножение, возможно сложение и проверка на чептность, что дает O(n). В итоге, временная асимптотика не превышает O(n^2)

№6

1)Храним: n - делимое (инициализируется значением числителя), m - константа равная знаметнателю, массив hvost[], в котором храним десятичный хвост числа, массив delimoe[] - в нем делимые (оба массива инициализируются нулями), nachalo - индекс элемента в массиве hvost, с которого начинается период десятичной дроби (в массивах индексы от 1).

```
if (n  > m)
        n = n % m
        length = 0
while (1 = 1){
for nachalo = 1 to length
        if (delimoe[nachalo] == n){
                for i = nachalo to length do
                        print(hvost[i])
                        stop
                }
length++
delimoe[length] = n
hvost[length] = (10*n)/m
n = (10*n) % m
}
```

На stop программа прекращается. Если периода нет, то программа не оставится. Корректность обьясняется тем, что именно так выполняется деление в столбик для нахождениия десятичных знаков после запятой и фактом, что если при делении у нас во второй раз получилось какое либо число, то при его делении мы получим те же частные, что и в первый раз, т.е. период, т.к. в конце этой повторяющейся последовательности будет опять это же число.
Предположим, что десятичная дробь имеет период длины n. Тогда для для того, чтобы найти его понадобиться операций порядка n^2. Можно уменьшить число операций до n, если учивая,что при делении на m можно получить всего m-1 ненулевых остатка, хранить их в массиве с отметкой "не посещен"либо номером итерации, когда был получен.

семинар-№5

Найти 3^{11}mod 107.
Вычисляем остаток по модулю для текущего делителя числа в процессе возведения, используя формулу $[a]_n * [b]_n = [a * b]_n$
3^{11} mod 107 $= (3^4)^2 * 3^3$ mod 107=$81^2 * 27$ mod 107 $= (-26)^2 * 27$ mod 107 = 676 mod 107 * 27 mod 107 = 34 * 27 mod 107 = 918 mod 107 = 62

Можно сделать это быстрее, используя бинарное возведение в степень. Так же будем вычислять остатки промежуточных значений по той же формуле.
Это будет выполняться следующим образом:

$$3^{11} = 3\left(3\left(3^2\right)^2\right)^2$$

Это шаги бинарного возведения в степень (демонстрация шагов алгоритма).
Вычисления произволим так (стрелка -> означает переход к следующему значению):

$$[1]_{107}->[1*3=3]_{107}->[3^2=9]_{107}->[9^2=81]_{107}=[-26]_{107}->[-26*3=-78]_{107}=$$
$$=[29]_{107}->[29^2=841]_{107}=[92]_{107}=[-15]_{107}->[-15*3=-45]_{107}=[62]_{107}$$

И непосредственно бинарное возведение (с помощью рекурсии возводим a в степнень n):

```
stepen ( a,  n) {
        if (n == 0)
                return 1;
        if (n % 2 == 1)
                return stepen (a, n-1) * a;
        else {
                 b = stepen (a, n/2);
                return b * b;
        }
}
```

Корректность алгоритма очевидна по примеру (т.е. он делает правильные действия при положительном показателе и при его занулении).
Он работает быстрее (за log n (n считаем целым)) по сравнению с привычным возведением в степень умножением на основание степени (действий n).
Но можно обойтись и без рекурсии. Для этого воспользуемся двоичным представлением показателя и запишем его как сумму степеней 2.

$$3^{11} = 3^{(1011)_2} = 3^{(2^3+2^1+2^0)} = 3^{2^3} * 3^2 * 3^1$$

На этом основан итеративный вариант бинарного возведения a в степень n

```
stepen ( a,  n) {
        stepen = 1;
       while (n)
```

```
                if (n & 1) {
                        stepen *= a;
                        --n;
                }
                else {
                        a *= a;
                        n >>= 1;
                }
        return stepen;
}
```

Его корректность очевидна на примере, а работает он с той же скоростью log n (n -целое) что и рекурсивый (в смысле число шагов то же, а системные вызовы и бинарные операции не учитываю).