HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

Algorithms and Data Structures

Laboratory work #2

Michael Kosyakov
Associate Professor

Denis Tarakanov
Assistant

Aglaya Iliina
Associate Professor

hduitmo.ads@yandex.ru

Classes plan

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

1. Previous homework problem #1005 “Stone Pile”
2. Problem #1401 “Gamers”
3. Task for homework
4. Explanation of test 1

Problem #1005 "Stone Pile"

- Link to the problem's description
  https://acm.timus.ru/problem.aspx?space=1&num=1005&locale=en

- You have a number of stones with known weights $w_1, \ldots, w_n$. Write a program that will rearrange the stones into two piles such that weight difference between the piles is minimal.
- Input contains the number of stones n ($1 \le n \le 20$) and weights of the stones $w_1, \ldots, w_n$ (integers, $1 \le w_i \le 100000$) delimited by white spaces.
- Your program should output a number representing the minimal possible weight difference between stone piles.

Problem #1005 "Stone Pile"

- 0-1 Knapsack problem
  - number x_i of copies of each kind of item restricted to zero or one
  - set of n items numbered from 1 up to n, each with a weight w_i and a value v_i
  - maximum weight capacity W

- Goal:
  - maximize $\sum_{i=1}^{n} v_i x_i$
  - subject to $\sum_{i=1}^{n} w_i x_i \le W$
  - where $x_i \in \{0, 1\}$

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Interpretation of problem as 0-1 Knapsack problem
  - $v_i = w_i$
  - maximize $\sum_{i=1}^{n} w_i x_i$
  - $\sum_{i=1}^{n} w_i x_i \leq \frac{\sum_{i=1}^{n} w_i}{2}$ - smaller pile

- Result: $\sum_{i=1}^{n} w_i - 2\sum_{i=1}^{n} w_i x_i$

Problem #1005 "Stone Pile"

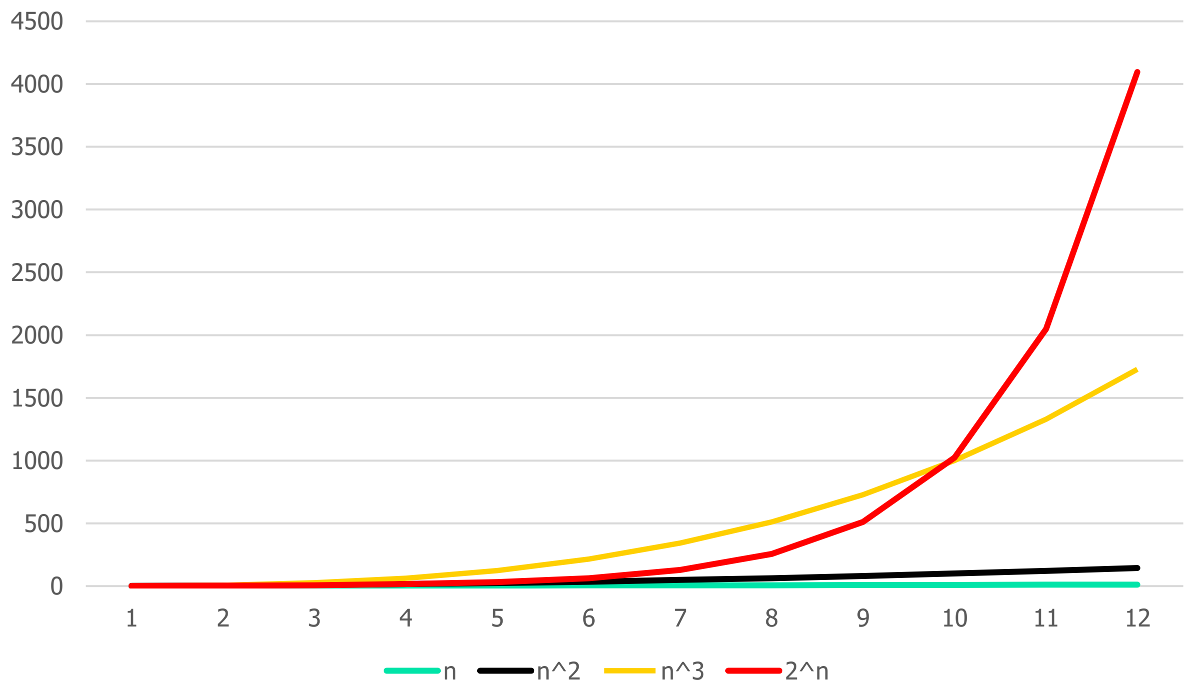
HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- NP-complete problem!

- Solutions of 0-1 Knapsack problem
  - Brute-force search
  - "Greedy" approximation algorithm
  - Dynamic programming

Problem #1005
"Stone Pile"

- Brute-force algorithm
  - Let's check all possible combinations of stones in piles!
  - Total 2^{n-1} different combinations!



Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Brute-force algorithm
  - Complexity is $O(2^n)$
  - With a sufficiently large number of objects, the problem becomes unsolvable by this method in an acceptable time

  - Good news: we have only 20 stones and 2^{19} combinations

Problem #1005 "Stone Pile"

- "Greedy" approximation algorithm
  - Sort all elements by value
  - Take elements with maximum values
  - In our case – take stones of maximum weights to the first pile until adding of any new stone will exceed half the sum of all weights
  - Idea from knapsack greedy algorithm

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Example 1
  - Let stones have weights {8, 11, 10, 4 , 6}
  - Sorted weights {11, 10, 8, 6, 4}
  - Sum of weights is 39, half sum is 19

First pile	-
Second pile	-

→

First pile	11
Second pile	-

→

First pile	11
Second pile	10

→

First pile	11, 8
Second pile	10

→

First pile	11, 8
Second pile	10, 6

→

First pile	11, 8
Second pile	10, 6, 4

- Result is |(11 + 8) – (10 + 6 + 4)| = 1

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Example 2
  - Let stones have weights {8, 13, 10, 6, 4, 10}
  - Sorted weights {13, 10, 10, 8, 6, 4}
  - Sum of weights is 51, half sum is 25

First pile	-
Second pile	-

First pile	13
Second pile	-

First pile	13, 10
Second pile	-

First pile	13, 10
Second pile	10

First pile	13, 10
Second pile	10, 8, 6, 4

  - Result is $|(13 + 10) - (10 + 8 + 6 + 4)| = 5$

Problem #1005 “Stone Pile”

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Example 2
  - Result = 5
  - Result is not optimal!

First pile	13, 8, 4
Second pile	10, 10, 6

- Result is $|(13 + 8 + 4) - (10 + 10 + 6)| = 1$

Problem #1005 "Stone Pile"

- Other "greedy" approach
  - Sort all elements by weight
  - Add next element to pile with less total weight
  - In example 1 {11, 10, 8, 6, 4} result is 3

First pile	**-**
Second pile	-

→

First pile	**11**
Second pile	-

→

First pile	**11**
Second pile	10

→

First pile	**11,**
Second pile	10, 8

→

First pile	**11, 6**
Second pile	10, 8

→

First pile	**11, 6, 4**
Second pile	10, 8

  - Result is not optimal!

Problem #1005 "Stone Pile"

- "Greedy" approximation algorithm
  - Algorithm calculates "approximate" result
  - Not applicable if accurate result is required
  - Very fast
    - Sorting: $O(n \log n)$
    - Iterate over sorted weights: $O(n)$
    - Total: $O(n \log n)$

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Dynamic programming
  - $W = \frac{\sum_{i=1}^{n} w_i}{2}$
  - Table t with size (N + 1) × (W + 1)
    - Starting from 0
  - Numerate all stones from 1 till N
  - t[i][j] – maximum possible sum of weights for stones with numbers { 1 .. i }, restricted by j
  - The result is t[N][W]
  - How to fill the table?

Problem #1005 "Stone Pile"

- Remember example 1

num	1	2	3	4	5
stone weight	8	11	10	4	6

- Table will have size 6 × 20
- t[4][5] = 4;   t[3][15] = 11;   t[4][16] = 11 + 4 = 15

- Let's iterate over table rows by i = { 0 .. N } and columns by w = { 0 .. W }
- Fill next element
  - t[0][w] = 0
  - t[i][w] = ?

Problem #1005 "Stone Pile"

- Case 1. $w_i > w$
  - No sense to use this stone
  - $t[i][w] = t[i-1][w]$
- Case 2. $w_i <= w$
  - What means to add stone?
  - Check maximum sum for all previous stones, if restrictions was $(w - w_i)$ – so we have enough place to add current stone
  - $t[i][w] = t[i-1][w - w_i] + w_i$

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Case 2. $w_i <= w$ (continue)
  - $t[i][w] = t[i - 1][w - w_i] + w_i$
  - It is possible, that to add current stone we remove stone with greater weight
  - Need compare this value with maximum without current stone
  - $t[i][w] = \max(t[i - 1][w - w_i] + w_i, t[i - 1][w])$
- By this 3 rules we can fill the table!

Problem #1005 "Stone Pile"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Dynamic programming
  - Time complexity is $O(n * W)$
    - It is called *pseudo-polynomial time*
  - Memory complexity is $O(n * W)$

  - Is $O(n * W)$ always better than $O(2^n)$?
  - In worst case for our problem:
    - Brute-force: $2^{19} \approx 500\,000$
    - Dynamic: $20 * \frac{100000}{2} * 20 \approx 2\,000\,000$

Problem #1401 "Gamers"

- Link to the problem's description https://acm.timus.ru/problem.aspx?space=1&num=1401&locale=en

- Mr. Chichikov argued with some blunderers that he would be able to prove that it is impossible to pave the 512 × 512 square checker-board with the figures:

```
X      XX      X      XX
XX     X      XX       X
```

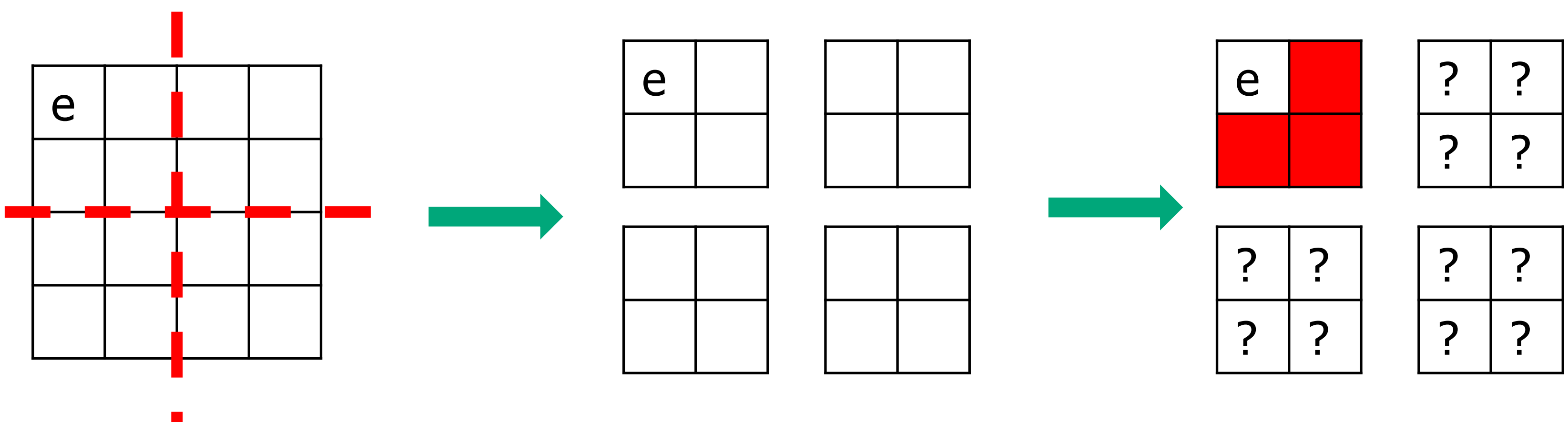
- Once one of those blunderers happened to be not so silly and he claimed that he was able to pave the 512 × 512 square checker-board without the upper right cell with those figures. Chichikov blurted out that he could pave any $2^n \times 2^n$ square checker-board without one arbitrary cell with those figures.

Problem #1401 "Gamers"

- Pave 2 × 2 square

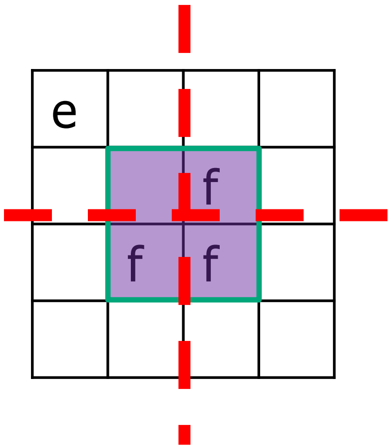
  - e – empty cell
  - Let's name the function to fill 2 × 2 square – pave2On2()
  - Arguments – coordinates of top left cell and empty cell

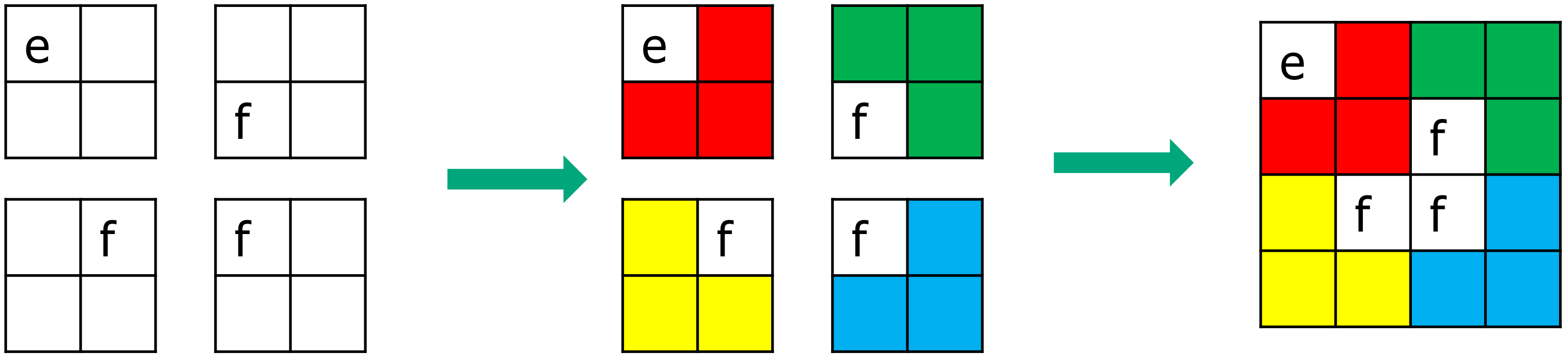
- Pave 4 ×4 square



Problem #1401 "Gamers"

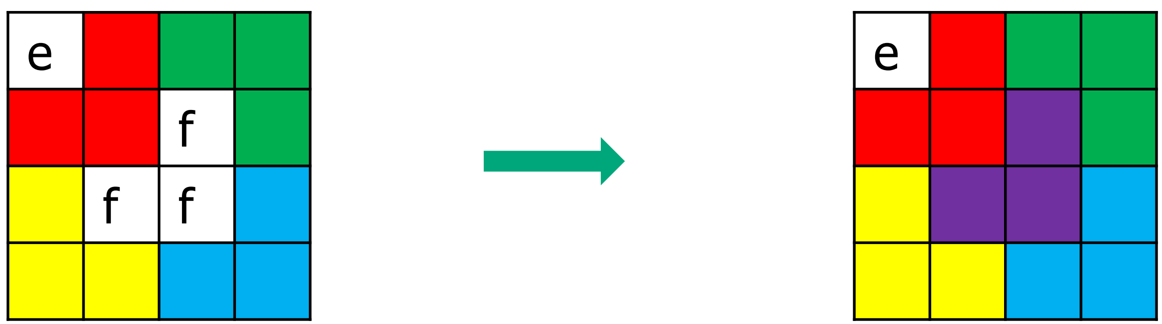
- Pave 4 × 4 square
  - "Middle" square
  - Fake empty cells
  - Provide fake empty cells to pave2On2()





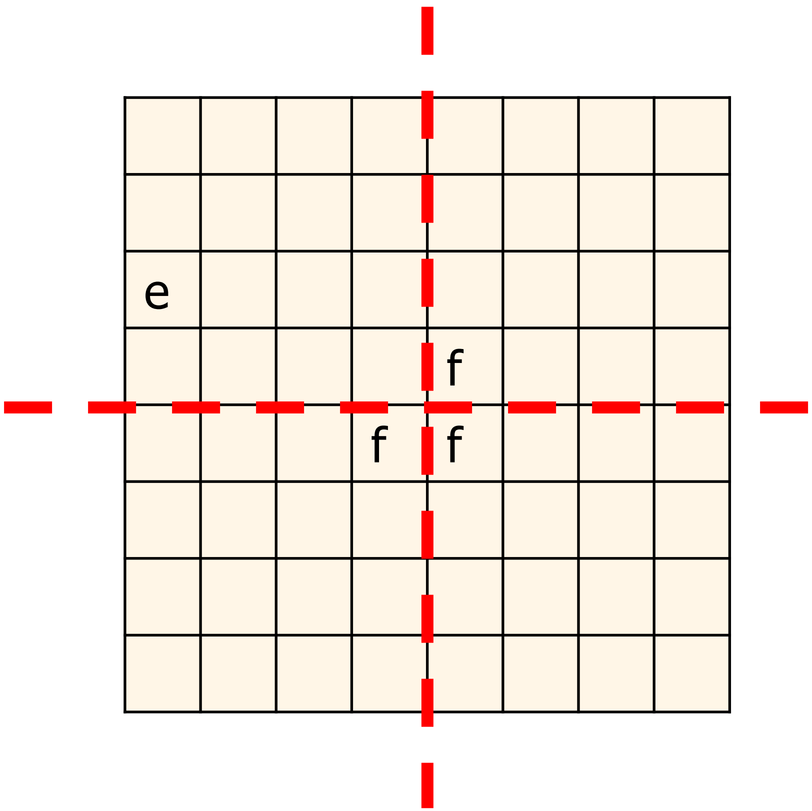
Problem #1401 "Gamers"

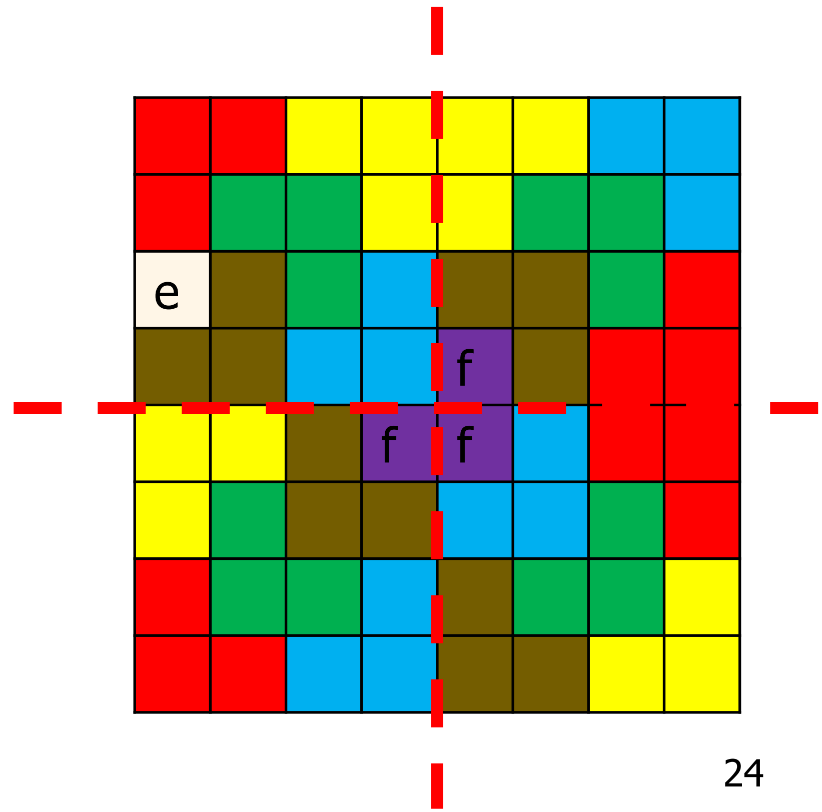
- Pave 4 × 4 square
  - Pave "middle" square
  - pave4On4()
  - Arguments are the same as for pave2On2()



Problem #1401 "Gamers"

- Pave 8 × 8 square
  - Let's use the same algorithm!





Problem #1401 "Gamers"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Pave $2^n \times 2^n$ square
  - Introduce paveSquare() recursive function
  - Let current size of square is $n \times n$
  - If $n = 2$ – easy to pave it based on empty cell
  - Otherwise – split square on 4 parts, each has size $\frac{n}{2} \times \frac{n}{2}$
  - If this part have empty cell – call paveSquare() for this part with real empty cell
  - Otherwise – choose fake empty cell in middle square and use to call paveSquare() for this part
  - Fill middle square

Problem #1401 "Gamers"

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Pave $2^n \times 2^n$ square
  - Complexity?
  - There are other ways to pave $2^n \times 2^n$ square – this algorithm is only example and prove that it is always possible
  - Feel free to use this algorithm of paving or choose another one

Mandatory task

1. Prepare source code to solve problem #1401 “Gamers”
   https://acm.timus.ru/problem.aspx?space=1&num=1401&locale=en
2. Pass tests on Timus system for this problem
   https://acm.timus.ru/submit.aspx?space=1&num=1401
3. Prepare a report with algorithm complexity and explanation
   Use template.docx to prepare report and send it to
   hduitmo.ads@yandex.ru with correct subject

Task for homework

You can solve following problem to get extra 2 points:

1. Problem #2025 “Line Fighting”
   https://acm.timus.ru/problem.aspx?space=1&num=2025&locale=en
   N.B. Report for this problem should contain explanation, why your approach to split fighters on teams is optimal

Explanation of test 1

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

- Task 1
  - Complexity of polynomic function
  - Simpler understanding is that for $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0,\ if\, a_k > 0$:
    - $p(n) = O(n^m),\ for\ m \geq k$
    - $p(n) = \Omega(n^m),\ for\ m \leq k$
    - $p(n) = \Theta(n^m),\ for\ m = k$

- Examples:
  - $f(N) = 15N^4 + 16N \ \Rightarrow\ f(N) = O(N^5)$
  - $f(N) = 75N + 8 \ \Rightarrow\ f(N) = \Theta(N)$
  - $f(N) = 12N^3 + 6N^2 + 3N + 1 \ \Rightarrow\ f(N) = \Omega(N)$

Explanation of test 1

- Task 2
  - Complexity of function in source code

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < l; j++) {
        // Complexity of function() is Θ(1)
        function(n, l);
    }
}
```

- Correct answer:
  - function() will be called l times for each start of internal loop
  - Internal loop will be started n times
  - Total n*m times
  - Complexity of function() is constant
  - $\Theta(n * l)$

Explanation of test 1

- Task 2
  - Complexity of function in source code

```
for (int i = 0; i < n / 2 + 2; i++) {
    // Complexity of function() is Θ(log n)
    function();
}
```

- Correct answer:
  - function() will be called (n / 2 + 2) times for each start of internal loop
  - Dependence is linear
  - Complexity of function() is $\Theta(\log n)$
  - $\Theta(n \log n)$

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

Explanation of test 1

- Task 2
  - Complexity of function in source code

```
for (int i = 0; i < n; i++) {
    i *= m;
}
```

- Correct answer:
  - How many iterations?
  - i will change in following way 0 => 1 => m => m^2 => ... => m^k >= n
  - k iterations, where k is solution of m^k >= n => $k \geq \log_m n$
  - $\Theta(\log_m n)$

HDU-ITMO Joint Institute
杭州电子科技大学 圣光机联合学院

Explanation of test 1

- Task 3
  - Time of recursion $T(N) = 4T(N/2) + 1$
  - Depth of recursion tree is $\log N$
  - On each level of recursion tree: $1 \Rightarrow 4 \Rightarrow 6 \Rightarrow 4^x$ elements
  - On the last level of tree:
    $x = \log N \Rightarrow 4^x = 4^{\log N} = 2^{\log N^2} = N^2$ elements
  - Geometric series from 1 to N^2 with step equal to 4, its sum
    $$\frac{1 * (4^{\log N+1} - 1)}{4-1} = \frac{4N^2 - 1}{3} = \Theta(N^2)$$
- Examples:
  - $T(N) = 4T\left(\frac{N}{2}\right) + 1 \Rightarrow T(N) = \Theta(N^2)$
  - $T(N) = 2T\left(\frac{N}{2}\right) + O(N) \Rightarrow T(N) = \Theta(\mathrm{N} \log N)$
  - $T(N) = 4T\left(\frac{N}{4}\right) + 1 \Rightarrow T(N) = \Theta(\mathrm{N})$
    - Note, depth of recursion tree in this case is $\log_4 N$

Explanation of test 1

- Task 4
- Stable sort
  - The order of equivalent elements is guaranteed to be preserved
  - Stability can be important for some cases, when equal elements are distinguishable
- William’s vs Floyd’s methods for building heap
  - Consecutive inserts vs sift down the roots of each subtree
  - Complexity O (N logN) vs O(N)
- Bottom-up approach for merge sort
  - Bottom-up: sort little groups in the whole array, then move to the larger groups
  - Doesn’t required recursion

Thank you!