**Lab sheet 02 b**

|  |  |
| --- | --- |
| **Example Task 1** | |
| **Task** | Write a Java method *sum(int n)* which takes an integer n and returns the sum of all positive integers less than or equal to n |
| Example input | 10 |
| Example output | 55 |
| Example input | 50 |
| Example output | 1275 |
| Solution |  |
| Tip | You can create your own java class to hold many such methods. |
| Analysis! | Show the elapsed time of your algorithm using several values of n and decide its time complexity using one of the seven functions. |

|  |  |
| --- | --- |
| **Example Task 2** | |
| **Task** | Write a Java method *sum\_const\_time(int n)* which take an integer n and returns the sum of all positive even integers less than or equal to n. your algorithm should perform in a constant time. Measure the elapsed time. |
| Example input | 10 |
| Example output | 30 |
| Example input | 50 |
| Example output | 650 |
| Solution |  |
| Tip | Remember Gauss’s formula (sum = n(n+1)/2)  evenSum = n(n+2)/4 |
| Analysis! | Show the elapsed time for your algorithm using several values of n. It should be constant or with slight variation. |
| Note | Don’t forget to measure the elapsed time. |

|  |  |
| --- | --- |
| **Example Task 3** | |
| **Task** | Write a Java method *concatDigits(int n)* which takes an integer and returns all the possible combinations (by concatenation) of the numbers between 0 and n. Remember that there are (n+1) \* (n+1) possible combinations for input n. |
| Example input | 2  0 1 2 |
| Example output | 22,21,20,12,11,10,02,01,00 |
| Solution |  |
| Analysis! | Show the elapsed time for your algorithm using several values of n. Analyse the time complexity of your algorithm and decide which of the seven function is applicable. |
| Note | Don’t forget to measure the elapsed time.  The use of String is slower than StringBuilder class. |

|  |  |
| --- | --- |
| **Example Task 4** | |
| **Task** | Write a Java method *plateNumber()* that generates all the possible four-digit plate numbers using the English alphabets A-Z (only capital letters) and Arabic numerals 0-9. |
| Example output | ABC1, A2DC, SHA1, …. |
| Solution |  |
| Analysis! | Analyse the time complexity of your algorithm and decide the function that best fits the time complexity |

|  |  |
| --- | --- |
| **Example Task 5 (Harder problem, needs analysis)** | |
| **Task** | Imagine there is a game played by n players who are assigned unique ids sorted in ascending order (player 1 has id 0, player 2 has id 1 ...player n has id n-1). Every time the game is played, the players will be split into two groups of equal size. One question will be asked, and whichever group answers correctly first, will remain in the game, but half of the players who do not answer will be sent out of the game. This process continues until one player stays and that player become a winner of the game.  Write a Java method called *eliminate-half()* that is used to remove half of the players until one winner remains. To decide the group to be removed, use a random number generator of either 0 or 1. If the random number is 0, the first group will be removed.  The game accepts the number of players as an input and displays the winner as an output. |
| Example | There are 10 players with id = 0,1,2, 3,…,9  Group 1: id 0-4  Group 2: id 5-9  Random number = 1  Group 2 is eliminated at this stage. Group 1 will remain in the next round |
| Solution |  |
| Analysis! | For the value of 100 players, how many steps are needed to find the winner.  Analyse the time complexity of your algorithm  What is the time complexity of the optimal algorithm? |