# Python - Amortized Analysis

Amortized analysis involves estimating the run time for the sequence of operations in a program without taking into consideration the span of the data distribution in the input values. A simple example is finding a value in a sorted list is quicker than in an unsorted list.

If the list is already sorted, it does not matter how distributed the data is. But of course the length of the list has an impact as it decides the number of steps the algorithm has to go through to get the final result.

So we see that if the initial cost of a single step of obtaining a sorted list is high, then the cost of subsequent steps of finding an element becomes considerably low. So Amortized analysis helps us find a bound on the worst-case running time for a sequence of operations. There are three approaches to amortized analysis.

- **Accounting Method** − This involves assigning a cost to each operation performed. If the actual operation finishes quicker than the assigned time then some positive credit is accumulated in the analysis.

  In the reverse scenario it will be negative credit. To keep track of these accumulated credits, we use a stack or tree data structure. The operations which are carried out early ( like sorting the list) have high amortized cost but the operations that are late in sequence have lower amortized cost as the accumulated credit is utilized. So the amortized cost is an upper bound of actual cost.

- **Potential Method** − In this method the saved credit is utilized for future operations as mathematical function of the state of the data structure. The evaluation of the mathematical function and the amortized cost should be equal. So when the actual cost is greater than amortized cost there is a decrease in potential and it is used utilized for future operations which are expensive.
- **Aggregate analysis** − In this method we estimate the upper bound on the total cost of n steps. The amortized cost is a simple division of total cost and the number of steps (n)..

**Print Page**