**Implementation of Amortized Analysis (Potential Method)**

**Theory**:

Amortized analysis is a method of analyzing algorithms that can help us determine an upper bound on the complexity of an algorithm. This is particularly useful when analyzing operations on data structures, when they involve slow, rarely occurring operations and fast, more common operations. With this disparity between each operations’ complexity, it is difficult to get a tight bound on the overall complexity of a sequence of operations using worst-case analysis. Amortized analysis provides us with a way of averaging the slow and fast operations together to obtain a tight upper bound on the overall algorithm runtime. Here we will consider a simplified version of the hash table problem, and show that a sequence of n insert operations has overall time O(n).

### Potential (Physicist's) Method

Suppose we can define a ***potential function*** Φ (read "Phi") on states of a data structure with the following properties:

* Φ(*h*0) = 0, where *h*0 is the initial state of the data structure.
* Φ(*ht*) ≥ 0 for all states *ht* of the data structure occurring during the course of the computation.

Intuitively, the potential function will keep track of the precharged time at any point in the computation. It measures how much saved-up time is available to pay for expensive operations. It is analogous to the bank balance in the banker's method. But interestingly, it depends only on the current state of the data structure, irrespective of the history of the computation that got it into that state.

We then define the ***amortized time*** of an operation as

| c + Φ(h') − Φ(h), |
| --- |

where *c* is the actual cost of the operation and *h* and *h*' are the states of the data structure before and after the operation, respectively. Thus the amortized time is the actual time plus the change in potential. Ideally, Φ should be defined so that the amortized time of each operation is small. Thus the change in potential should be positive for low-cost operations and negative for high-cost operations.

Now consider a sequence of *n* operations taking actual times *c*0, *c*1, *c*2, ..., *cn*−1 and producing data structures *h*1, *h*2, ..., *hn* starting from *h*0. The total amortized time is the sum of the individual amortized times:

(*c*0 + Φ(*h*1) − Φ(*h*0)) + (*c*1 + Φ(*h*2) − Φ(*h*1)) + ... + (*cn*−1 + Φ(*hn*) − Φ(*hn*−1))

= *c*0 + *c*1 + ... + *cn*−1 + Φ(*hn*) − Φ(*h*0)

= *c*0 + *c*1 + ... + *cn*−1 + Φ(*hn*).

Therefore the amortized time for a sequence of operations overestimates of the actual time by Φ(*hn*), which by assumption is always positive. Thus the total amortized time is always an upper bound on the actual time.

Code:

| # include <iostream> # include <bits/stdc++.h> using namespace std;  void print(int \*arr, int n){  for(int i=0; i<n; i++){  cout<<arr[i]<<" ";  }  cout<<endl; }  int potential(int count, int size){  return (2\*count - size); }  int main(){  int size = 1;  int count = 0;   int arr[size];  int \*p = arr;   while(1){  int n;  cout<<"Enter the number you want to add in the dynamic array: ";  cin>>n;  if(count<size){  \*(p+count) = n;  count+=1;  cout<<"Potential: "<<potential(count, size)<<endl;  print(p,count);  }else{  //double  cout<<"DOUBLE"<<endl;  int \*new\_arr = new int[size\*2];  for(int i=0; i<count; i++) new\_arr[i] = \*(p+i);  size\*=2;  p = new\_arr;  \*(p+count) = n;  cout<<"Potential: "<<potential(count, size)<<endl;  count+=1;  print(p,count);  }  cout<<endl;  }   return 0; } |
| --- |

Output:

| Enter the number you want to add in the dynamic array: 5 Potential: 1 5   Enter the number you want to add in the dynamic array: 10 DOUBLE Potential: 0 5 10   Enter the number you want to add in the dynamic array: 2 DOUBLE Potential: 0 5 10 2   Enter the number you want to add in the dynamic array: 19 Potential: 4 5 10 2 19   Enter the number you want to add in the dynamic array: 5 DOUBLE Potential: 0 5 10 2 19 5   Enter the number you want to add in the dynamic array: 23 Potential: 4 5 10 2 19 5 23 |
| --- |

**Observations**:

The amortized time for a sequence of operations overestimates of the actual time by Φ(hn), which by assumption is always positive. Thus the total amortized time is always an upper bound on the actual time.

**Conclusion**:

The key to amortized analysis with the physicist's method is to define the right potential function. The potential function needs to save up enough time to be used later when it is needed. But it cannot save so much time that it causes the amortized time of the current operation to be too high and the sequence of n insert operations has overall time O(n).

**References**:

<https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec21-amortized/lec21.html>