• Basic Concept
• Example: Dynamic Array
• Techniques: Aggregate, accounting (charging), potential

- “amortize”: paying off debt/mortgage
- Idea: Certain steps in algorithm may be very expensive
  if these steps don’t happen often, total running time is bounded.
- Problem: dynamic array
  recall: java has “arraylist”, vector is a growing array
  vector supports “append” operation: add 1 element to end of vec.
  Goal: do not want to waste a lot of space
  make sure append operation is not very slow

- Solution: initially 1 element, space of size 1
  append
  
  ```
  if length = 2^i (length = 1, 2, 4, 8, ...) 
  allocate new space of size 2^{i+1} 
  copy the current 2^i elements to the new space 
  put new element in 2^{i+1} location 
  free the old space 
  else 
  put new element into first free space.
  ```

- Example

```
length 4 \rightarrow 5
```

- Space: If there are n elements in array, it has \( \leq 2n \) space.
- Running time: append operation can take \( O(n) \) time.
  
  naive analysis: if we do \( n \) append operations
it can take $O(n^2)$ time.

- Aggregate: take the sum of running times
  
  step $2^i$ takes $2^i$ operations
  other steps take 1 operation
  
  $T(n) = \sum_{i=0}^{\log_2 n} 2^i + \sum_{i=1}^{n} 1$

  "heavy" operations
  "light" operations

  $\leq 2n + n \leq 3n$

  simple, but not very general.

- Accounting (charging)
  
  idea: save "money" for light operations, pay money for heavy operations
  
  observation: "heavy" operation comes at time $2^i$

  before that, $2^{i-1}, 2^{i-2}, \ldots, 2^1$ are light operations

  $2^{i-1} - 1$ light operations

  if we save 2 time units per light operation, can "charge" $2^{i-2}$ to these light operation, and the remaining 2 is charged to the current operation.

  - For every operation: time paid + time saved $\leq 3$

  Total runtime $\leq 3n$

- Potential argument

  Keep a potential function $\Phi$, $\Phi \geq 0$

  amortized cost for an operation = actual cost - current potential

  + new potential.

  $= \text{actual cost} - (\Phi_{\text{current}} - \Phi_{\text{new}})$

  $\Phi_i = 2^n - n$
\[ \Phi = 2^n - m \]

\[ \Phi_{\text{current}} \]

\[ \Phi_{\text{new}} = 2 \]

**“Heavy” step:** before \( n = 2^i \), \( m = 2^i \)  
after \( n = 2^{i+1} \), \( m = 2^{i+1} \)  
actual cost = \( 2^i \)  
amortized cost = \( 2 \)

**“Light” step:** before \( n = m \)  
after \( n+1 = m \)  
actual cost = \( 1 \)  
amortized cost = \( 3 \)

**Claim:** \[ \sum \text{actual cost} = \sum \text{amortized cost} + \Phi_{\text{init}} - \Phi_{\text{end}} \]

**Proof:** Suppose alg had \( k \) steps.

Let \( \Phi_i \) be the potential function after step \( i \)

\[ \text{for step } i \text{ actual cost} = \text{ amortized cost} + (\Phi_{i-1} - \Phi_i) \]

\[ = \sum \text{actual cost} = \sum \text{amortized cost} + \Phi_0 - \Phi_1 + \Phi_1 - \Phi_2 + \cdots + \Phi_k \]

\[ = \sum \text{amortized cost} + \Phi_0 - \Phi_k \]

therefore \[ \sum \text{actual cost} = \sum \text{amortized cost} + \Phi_{\text{init}} - \Phi_{\text{end}} \]

For dynamic array \[ \Phi_{\text{init}} = 1 \]

\[ \sum \text{actual cost} \leq \sum \text{amortized cost} + 1 = O(n) \]

\[ \text{because } \Phi_{\text{end}} \geq 0 \]

- Simple examples we’ve seen that uses the idea of “amortize”
  - DFS \( O(n+m) \)
  - merge sort \( O(n) \)