Data Structures & Programming

Complexity Analysis Examples

Golnar Sheikhshab

Prefix averages

given an array X storing n numbers, we want to compute an array A such that A[i] is the average of elements X[0],...,X[i], for i = 0,...,n−1, that is,

$$A[i] = \frac{\sum_{j=0}^{i} X[j]}{i+1}$$

Prefix averages - a ***Quadratic*** Solution

Algorithm prefixAverages1(X):

  Input: An n-element array X of numbers.

  Output: An n-element array A of numbers such that $A[i]$ is the average of elements $X[0],\ldots,X[i]$.

Let A be an array of n numbers.

for $i \leftarrow 0$ **to** $n-1$ **do**

  $a \leftarrow 0$

  for $j \leftarrow 0$ **to** i **do**

    $a \leftarrow a + X[j]$

  $A[i] \leftarrow a/(i+1)$

return array A

Prefix averages - a ***Linear*** Solution

Algorithm prefixAverages2(X):

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[i]$ is the average of elements $X[0],\ldots,X[i]$.

Let A be an array of n numbers.

$s \leftarrow 0$

for $i \leftarrow 0$ **to** $n-1$ **do**

  $s \leftarrow s + X[i]$

  $A[i] \leftarrow s/(i+1)$

return array A

Two recursive computation of power

Linear time complexity

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, n-1) & \text{otherwise} \end{cases}$$

Logarithmic time complexity

$$p(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot p(x, (n-1)/2)^2 & \text{if } n > 0 \text{ is odd} \\ p(x, n/2)^2 & \text{if } n > 0 \text{ is even} \end{cases}$$

Three-way set disjoint

```
bool areDisjoint(const vector<int>& a, const vector<int>& b,
                 const vector<int>& c) {
  for (int i = 0; i < a.size(); i++)
    for (int j = 0; j < b.size(); j++)
      for (int k = 0; k < c.size(); k++)
        if ((a[i] == b[j]) && (b[j] == c[k])) return false;
  return true;
}
```

Cubic time complexity

Element uniqueness problem

Recursive solution

```
bool isUnique(const vector<int>& arr, int start, int end) {
  if (start >= end) return true;
  if (!isUnique(arr, start, end-1))
    return false;
  if (!isUnique(arr, start+1, end))
    return false;
  return (arr[start] != arr[end]);
}
```

Element uniqueness problem

Iterative solution

```
bool isUniqueLoop(const vector<int>& arr, int start, int end) {
  if (start >= end) return true;
  for (int i = start; i < end; i++)
    for (int j = i+1; j <= end; j++)
      if (arr[i] == arr[j]) return false;
  return true;
}
```

Element uniqueness problem

Sort-based solution

```
bool isUniqueSort(const vector<int>& arr, int start, int end) {
  if (start >= end) return true;
  vector<int> buf(arr);                       // duplicate copy of arr
  sort(buf.begin()+start, buf.begin()+end);   // sort the subarray
  for (int i = start; i < end; i++)           // check for duplicates
    if (buf[i] == buf[i+1]) return false;
  return true;
}
```

Some algorithms with complexity O(1)

- Adding an item in front of a linked list

```
void intSLinkedList::addFront(const int& e) {
    intSNode* v = new intSNode;
    v->elem = e;
    v->next = head;
    head = v;
}
```

- Adding an item at the end of an array

```
void add(const E& new_entry){
        listArray_[n_] = new_entry;
        n_++;
}
```

Analyzing an algorithm even further

In this function, how many times does the value of max change?

```
int findMax(const vector<int>& arr) {
  int max = arr[0];
  for (int i = 1; i < arr.size(); i++) {
    if (max < arr[i]) max = arr[i];
  }
  return max;
}
```

Worst case scenario: n-1 times, which is O(n)

Average case scenario: $H_n = \sum_{i=1}^{n} 1/i$ times which is O(log n)

Reading material

Section 4.2