PIC 10B Discussion
Week 7
Professor: Michael Lindstrom
TA: Thomas Tu
Telescoping Series

• Scenario: I give you 1 apple
• I take it away, then give you 2 apples
• I take them both away, then give you 3.....
• I’ve just given you 20 apples, how many do you have now?
• $1 + (2-1) + (3-2) + (4-3) + ..... + (20-19)$
• Summation notation
Recursion

- Natural consequence of function structure
- Hard part: making sure what you want is what is happening
- Simple example: factorial
  - Note: you shouldn’t ever actually compute factorial this way....
- Common issue: stack overflow
Why Use Recursion?

- If it makes your code easier to write
- If it makes your code easier to read
- If it makes your code easier to understand

- Note: In C++ and many other languages, recursion is less efficient than iteration
- This depends on implementation, though
- It is also a theorem that you can always use iteration instead
  - However, theorems aren’t concerned with practicality
Costing Algorithms

• Big-O notation: the approximate scaling of an algorithm, as the input size goes to infinity
• Useful for telling if one algorithm is better or worse than another without having to test on a particular machine
• Format: $O(1)$, $O(n)$, $O(n^2)$, $O(n \log(n))$
• Note: based on scaling, not exact numbers
  • It’s okay to ignore small terms
Recursive Bubble Sort

```cpp
void bubSort(std::vector& v, size_t beg, size_t end) {
    if ( (end > v.size() - 1) || (beg > end) )
        throw std::exception("invalid input range");
    if ( beg == end) return; // done if this is the case
    for (size_t i = beg; i < end; ++i) // loop over adjacent pairs
        if (v[i] > v[i + 1]) // swap values if necessary
            std::swap(v[i], v[i + 1]);
    bubSort(v, beg, end-1); // sort over smaller range with last item in place
}
```
Recursive Bubble Sort

bubSort(vector<int> v, size_t beg, size_t end) {
    // check beg==end: constant time
    for(size_t i=beg;i<end;i++) //repeat the following (end-beg) times
        // check if v[i]>v[i+1]: constant time
        // if so, do a swap: constant time
        bubSort(v, beg, end-1)
}

• What is the order of bubble sort?
  • you should remember from yesterday...
Recursive Bubble Sort

• Let \( f(\text{beg}, \text{end}) \) be the time taken by bubble sort
• \( f(\text{beg}, \text{end}) = c_1 + (\text{end}-\text{beg}) \times c_2 + f(\text{beg}, \text{end}-1) \)
• Let \( n \) be \( \text{end}-\text{beg} \)
• \( f(n) = n \times c_2 + f(n-1) = 2n \times c_2 + f(n-2) = \ldots \)
  • Can ignore \( c_1 \) term, since \( n \times c_2 \) is so much larger than \( c_1 \)
• Summation:
  • \( f(n)-f(n-1) = n \times c_2 \)
  • \( \sum_{1}^{n} f(k) - f(k - 1) = f(n) - f(0) = f(n) \)
  • \( \sum_{1}^{n} k \times c_2 = c_2 \sum_{1}^{n} k = c_2 \frac{k(k+1)}{2} \) is \( O(k^2) \)
Quick Sort

• Take the last element and call it the pivot
• Go through the list from start to end
  • If an element is less than the pivot, swap it with one you know is greater
  • End result: half the list is smaller and half is bigger
  • Sort both halves (using quicksort)
  • Put the pivot in the middle, and now it’s sorted
Quick Sort

```cpp
void quickSort(vector<int>& v, size_t beg, size_t end) {
    if (beg >= end) return; // done if this is the case
    int pivot = v[end]; // choose the pivot arbitrarily
    int low = beg; // location of the last known element lower than the pivot
    if (v[low] > pivot) { std::swap(v[low], v[end]); pivot = v[end]; } // ensure first element is lower than pivot
    for (size_t i = beg+1; i < end; i++) {
        if (v[i] < pivot) { // found a new lower element, stick it in the left half
            low++;
            std::swap(v[low], v[i]);
        }
    }
    low++;
    std::swap(v[low], v[end]);
    quickSort(v, beg, low-1); // sort the left half
    quickSort(v, low + 1, end); // sort the right half
}
```
Quick Sort

quickSort(v, beg, end):
// constant time: check ends, find pivot, check pivot
for (size_t i = beg+1; i < end; i++) // repeat end-(beg+1) times
    // constant time: check the condition, perform swap
// constant time: move the pivot into the right place
quickSort(v, beg, low-1)
quickSort(v, low + 1, end)
Quick Sort

• $f(beg, end) = c_1 + (end - beg - 1) \times c_2 + f(beg, low - 1) + f(low + 1, end)$
• Assumption: low is halfway between beg and end
• $f(n) = c_1 + (n - 1) \times c_2 + f(n/2) + f(n/2)$
• $f(n) = n \times c_2 + 2 \times f(n/2) = n \times c_2 + 2 \times n \times c_2 + 2 \times 2 \times f(n/4) = \ldots$
• What is the order of this method?
  • You can suppose $n = 2^k$, where $k = \log(n)$
Sorting Visualization

• YouTube video: 15 Sorting Algorithms in 6 Minutes