School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

MSSP + Sorting

Previously on ADDS

- Binary search
- Benefits:
  - halve the search space every time
  - don't have to search every element
- Complexity – O(log n)
  - log with base 2 is usually denoted by lg or log_2. The default base of log is 10
  - But we usually mean base 2 in computer science, and it does not make a difference in terms of Big O notation.
- Sorted data can be searched faster
- Sort once, search a lot

Overview

- See one more problem with different solutions (algorithms)
- Start the topic of Sorting

Example

- Maximum Subsequence Sum Problem
- Given (possibly negative) integers A_1, A_2 … , A_n, the target of the problem is to find the maximum value of
$$\sum_{k=i}^{j} A_k$$
, where $i, j \epsilon [1, n]$.

- Example: -1, 2, 3, 6, -12, 13   -1, 2, 3, 6, -12, 13   13
- -1, 2, 3, 6, -8, 13   -1, 2, 3, 6, -8, 13   16

- There are many different algorithms to solve it and the performance of these algorithms varies significantly.

Algorithm1

```
//input: arr
Int maxsum=0
for(I=0 to arr.size)
        for(j=I to arr.size)
        {
                int sum=0;
                for(k=I to j)
                        sum+=arr[k]
                if(sum> maxsum)
                        maxsum=sum
        }
return maxsum
```

$O(n^3)$

Example -MSSP

- Algorithm 2
- $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$

O(n^2)

```
int maxSubSum2(int a[], int size){

  int maxSum = 0;
  for(int i=0; i<size; i++){
    int sum = 0;
    for(int j=i; j<size; j++){
      sum+= a[j];
      if(sum>maxSum)
        maxSum = sum;
    }
  }
  return maxSum;
}
```

Divide and conquer strategy

- Divide – split the problem into two roughly equal subproblems which are then solved recursively
- Conquer – patch together the two solutions and possibly do a small amount of additional work to arrive at a solution to the whole problem.

- Algorithm 3 for MSSP
  - Can we use divide and conquer?
  - Yes. With complexity O(n log n)

Divide and Conquer

```
int maxSubArray(int [] A, int start, int end){
if(start==end){
return A[start];
}
int mid = start + (end-start)/2;
int leftMaxSum = maxSubArray(A, start, mid);
int rightMaxSum = maxSubArray(A, mid+1, end);

int sum = 0;
int leftMidMax =0;
for (int i = mid; i >=start ; i--) {
sum += A[i];
if(sum>leftMidMax)
leftMidMax = sum;
}
sum = 0;
int rightMidMax =0;
for (int i = mid+1; i <=end ; i++) {
sum += A[i];
if(sum>rightMidMax)
rightMidMax = sum;
}
int centerSum = leftMidMax + rightMidMax;
return Math.max(centerSum, Math.max(leftMaxSum, rightMaxSum));
}
```

Source: Tutorialhorizon

Master Theorem

Theorem (master theorem, simple form):

For positive constants a, b, c, and d, and $n = b^k$ for some integer k, consider the recurrence

$$r(n) = \begin{cases} a, & \text{if } n = 1 \\ cn + d \cdot r(n/b), & \text{if } n \geq 2 \end{cases}$$

then

$$r(n) = \begin{cases} \Theta(n), & \text{if } d < b \\ \Theta(n \log n), & \text{if } d = b \\ \Theta(n^{\log_b d}) & \text{if } d > b. \end{cases}$$

Example Master Theorem

$$T(n) \leq \begin{cases} 1, & \text{if } n = 1 \\ 4 \cdot T(\lceil n/2 \rceil) + 3 \cdot 2 \cdot n, & \text{if } n \geq 2 \end{cases}$$

Consider $n = 2^k$

$a = 1$, $b = 2$, $c = 6$, and $d = 4$

$d > b : \Theta(n^{\log_b d}) = \Theta(n^{\log_2 d}) = \Theta(n^2)$

Kadane's Algorithm

Kadane's algorithm uses dynamic programming.

maxSum for array consisting of first element a[1] is a[1].

maxSum for subarray a[1, …,i] is computed based on maxSum for a[1, …, i-1] using

maxSum[i] = max{maxSum[i-1], maxSum ending at i-1 + a[i]}

maxSum ending at i-1 is 0 when not using any element of a[1, …,i-1]
Use this if using elements ending at position i-1 would become negative.

Example - MSSP

- Algorithm 4 with complexity in O(n)

Kadane's Algorithm

```
int maxSubSum4(int a[], int size){

  int maxSum, sum = 0;

  for(int j=0; j<size; j++){
    sum += a[j];
    if(sum>maxSum)
      maxSum = sum;
    else if(sum<0)
      sum = 0;
  }
  return maxSum;
}
```

Sorting Algorithms

- Insertion Sort
- Selection Sort
- Bubble Sort
- Quicksort
- Merge Sort
- Heapsort (later; if we have enough time)

- Bucket sort

Insertion Sort

```
function insertionsort(list){
 for i = 1 to length(list)
    x = list[i]
    j = i - 1
    for j = i-1 to 0
       if A[j] > x
          A[j+1] ← A[j]
       else
         break for loop              // end inner for loop

    A[j+1] = x

}
```

- Insertion sort is a simple sorting algorithm

- Complexity
  - worst-case $O(n^2)$
  - average-case $O(n^2)$
  - best-case $O(n)$

Selection Sort

- In selection sort, the list is divided into two part:
  - Sorted part (considering all elements)
  - Unsorted part

Input: list
For i=0 to n
{
    j= find the index with min value among list[i] to list[n]
    swap list[i] and list[j]
}

- Complexity
  - worst-case $O(n^2)$
  - average-case $O(n^2)$
  - best-case $O(n^2)$

THE UNIVERSITY of ADELAIDE