School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Impact of Algorithm Design - MSSP

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)

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

$$\sum_{k=i}^{j} A_k \quad \text{where } i, j \in [1, n].$$

- 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.

Algorithm 1

```
//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
```

Algorithm 2

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

```
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?
  - What would be the complexity?

Algorithm 3: 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 constants $a \geq 1$, $b \geq 2$, $d \geq 0$ and $f(n) \in \Theta(n^d)$, consider the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b^a}) & \text{if } a > b^d \end{cases}$$

Algorithm 4

- Kadane's Algorithm, complexity in O(n)

```
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;
}
```