make history.

THE UNIVERSITY of ADELAIDE

Other recursive algorithms

Dr. Anna Kalenkova

Binary search

Benefits:

- halve the search space every time
- don’t have to analyse any element
- complexity O(log n)
- sorted data can be searched faster
- sort once, search a lot

Recursive Fibonacci

```
int fib(int n) {
    if(n<=1)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

$$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Recursive Fibonacci is $O(2^n), \Omega(2^{n/2})$

The greatest common divisor (Euclid's algorithm)

```
int recursiveGCD (int a, int b) {
    if(b==0) return a;
    return recursiveGCD(b, a%b);
}
int gcd(int a, int b) {
    while(b!=0) {
        int reminder = a%b;
        a = b;
        b = reminder;
    }
    return a; }
```

THE UNIVERSITY
of ADELAIDE

The greatest common divisor (Euclid's algorithm)

```
int gcd(int a, int b) {
    while(b!=0) {
        int reminder = a%b;
        a = b;
        b = reminder;
    }
    return a; }
```

Values a and b are monotonically decreasing;

After the first two iterations a <= min {a, b};

After any two iterations the value of a is at most half of what it has been before;

Hence, the tome complexity is O(min{a,b}).

THE UNIVERSITY of ADELAIDE

Maximum subsequence sub-problem

Find subsequence with maximum sum:

-1, 2, 3, 6, -12, 13 → 13

- 1, 2 ,3, 6, -8, 13 → 13

There are different algorithms to solve this problem and their performance varies significantly.

Algorithm 1

```
for(size_t i = 0; i < s.size(); i++) {
    for (size_t j = i; j < s.size(); i++) {
        int sum = 0;
        for (size_t k = i; k <= j; k++) {
            sum += s.at(k);
        }
        if (sum > max) { max = sum;}
    }
}
return max;
```

Algorithm 2

```
for(size_t i = 0; i < s.size(); i++) {
    int sum = 0;
    for (size_t j = i; j < s.size(); i++) {
        sum += s.at(j);
        if (sum > max) { max = sum;}
    }
}
return max;
```

Algorithm 3: Divide and conquer

```
int maxSubArray(vector<int> s, int start, int end) {
if (start == end) {return s.at(0);}
int mid = start + (end-start)/2;
int leftMaxSum = maxSubArray(s, start, mid); int rightMaxSum = maxSubArray(s, mid + 1, end);
int sum = 0; int leftMidMax = 0;
for (int i = mid; i >=start; i--) {
     sum+=s.at(i);
     if (sum > leftMidMax) {
         leftMidMax = sum;
        }
}
sum = 0; int rightMidMax = 0;
for (int i = mid + 1; i <=end; i++) {
….
}
int centerSum = leftMidMax + rightMidMax;
return max(centerSum, max(leftMaxSum, rightMaxSum));
}
```

THE UNIVERSITY of ADELAIDE

Algorithm 4: Kadan’s algorithm

```
int maxSubArray(vector<int> s) {
    int sum, maxSum = 0;
    for (int i = 0; i < s.size(); i++) {
        sum+=s.at(i);
        if (sum > maxSum) {
            maxSum = sum;
        } else if (sum < 0) {
        sum =0;
     } return maxSum;
  }
```