Estruturas de Dados e Algoritmos

Algoritmos — conceitos básicos

Algoritmo:

Sequência de passos simples, claramente especificada, para resolver um determinado problema.
Uma receita.

Problemas e Algoritmos — exemplos

Problema 1:

Dada uma sequência de **N** números, indicar o k-ésimo maior. Designado por problema da selecção (*selection problem*)

Solução "imediata"

Solução 1:

Ler os N números guardando-os num vector. Ordenar o vector por ordem decrescente usando por exemplo o *bubblesort*. Devolver o número na k-ésima posição do vector.

Melhorando a Solução "imediata"

Solução 2:

Ler os k primeiros números guardando-os num vector de tamanho k. Ordenar o vector por ordem decrescente. Ler os restantes números e inserir ordenadamente no vector só se for maior que o número na última posição. Se inserir um elemento deitar fora o último. No final devolver o último elemento do vector.

Problemas e Soluções

- Para uma simulação de 1 milhão de números gerados aleatoreamente e k = 500 mil, os 2 algoritmos demorariam alguns dias e dar a resposta. O que não é aceitável.
- Existe uma solução 3 (mais elaborada) que demora alguns segundos.
- **Conclusão:** Soluções simples, fáceis de implementar nem sempre são as mais eficientes e até execuíveis!!

Análise de Algoritmos

- Tempo e Espaço
- Exemplos
- Notação O()

Análise de Algoritmos

- Como obter a solução de um problema?
  - Especificação do problema: descrição da relação entre os valores de entrada e os de saída
  - Algoritmo: procedimento computacional que toma os valores de entrada e os transforma nos de saída
- Análise de algoritmos
  - Recursos necessários para executar o procedimento respectivo
    - Tempo de processamento
    - Espaço de armazenamento
  - Avaliação independente da máquina
    - Baseada no crescimento dos recursos com a entrada

Análise de Algoritmos

Considerações gerais

- Geralmente compromisso entre Espaço ocupado/Tempo de execução
  ex: se quero mais rápido uso estruturas de dados auxiliares
- Estruturas de Dados e Algoritmos essencialmente para problemas complexos. Para problemas simples qq serve
  ex: espaço de soluções pequeno uso "gerar e testar"
- Perspectiva pragmática:
  se existe feito então **use**
  senão se quase existe feito então **adapte**
  senão **desenvolva**
- Adopte sempre a solução mais simples. Não complicar desnecessariamente

Crescimento de funções

- Comparar crescimento
  - Comparação de funções em pontos particulares: muito dependente dos coeficientes
  - Comparação relevante: taxas de crescimento
- Avaliar taxa de crescimento
  - em funções com vários termos o crescimento é determinado pelo termo de crescimento mais rápido
  - Coeficientes constantes influenciam o andamento inicial

Função	**Designação**
c	Constante
log N	Logaritmo
$\log^2$ N	Logaritmo Quadrado
N	Linear
N log N	N log N
N^2	Quadrática
N^3	Cúbica
2^N	Exponencial

Notação O(*)

Notação para o crescimento relativo de funções

- $T(n) = O(f(n))$ **se** existem constantes **c** e **n'** tais que $T(n) \leq c\ f(n)$ para $n \geq$ **n'**
- $T(n) = \Omega(f(n))$ **se** existem constantes **c** e **n'** tais que $T(n) \geq c\ f(n)$ para $n \geq$ **n'**
- $T(n) = \Theta(f(n))$ **se e só se** $T(n) = O(f(n))$ e $T(n) = \Omega(f(n))$
- $T(n) = o(f(n))$ **se** $T(n) = O(f(n))$ e $T(n) \neq \Theta(f(n))$

Crescimento de função composta

Regras

1. se T1(n) = O(f(n)) e T2(n) = O(g(n))
   - T1(n) + T2(n) = max(O(f(n)), O(g(n)))
   - T1(n) * T2(n) = O(f(n) * g(n))
2. se T(x) é um polinómio de grau n, T(x) = Θ (x^n)
3. $\log^k$ n = O(n), para qualquer k (logaritmos crescem devagar)

- não usar constantes ou termos de ordem inferior dentro de um O(*)
- crescimento relativo pode ser determinado com $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

Subsequência de Soma Máxima

Problema:

Dada uma sequência A_1, A_2, ..., A_N de números inteiros (possivelmente com negativos), encontrar o valor máximo para $\sum_{k=i}^{j} A_k$

Exemplo:

Dada uma sequência: -2, 11, -4, 13, -5, -2
Resposta: 20 (A_2 a A_4)

Subsequência de Soma Máxima

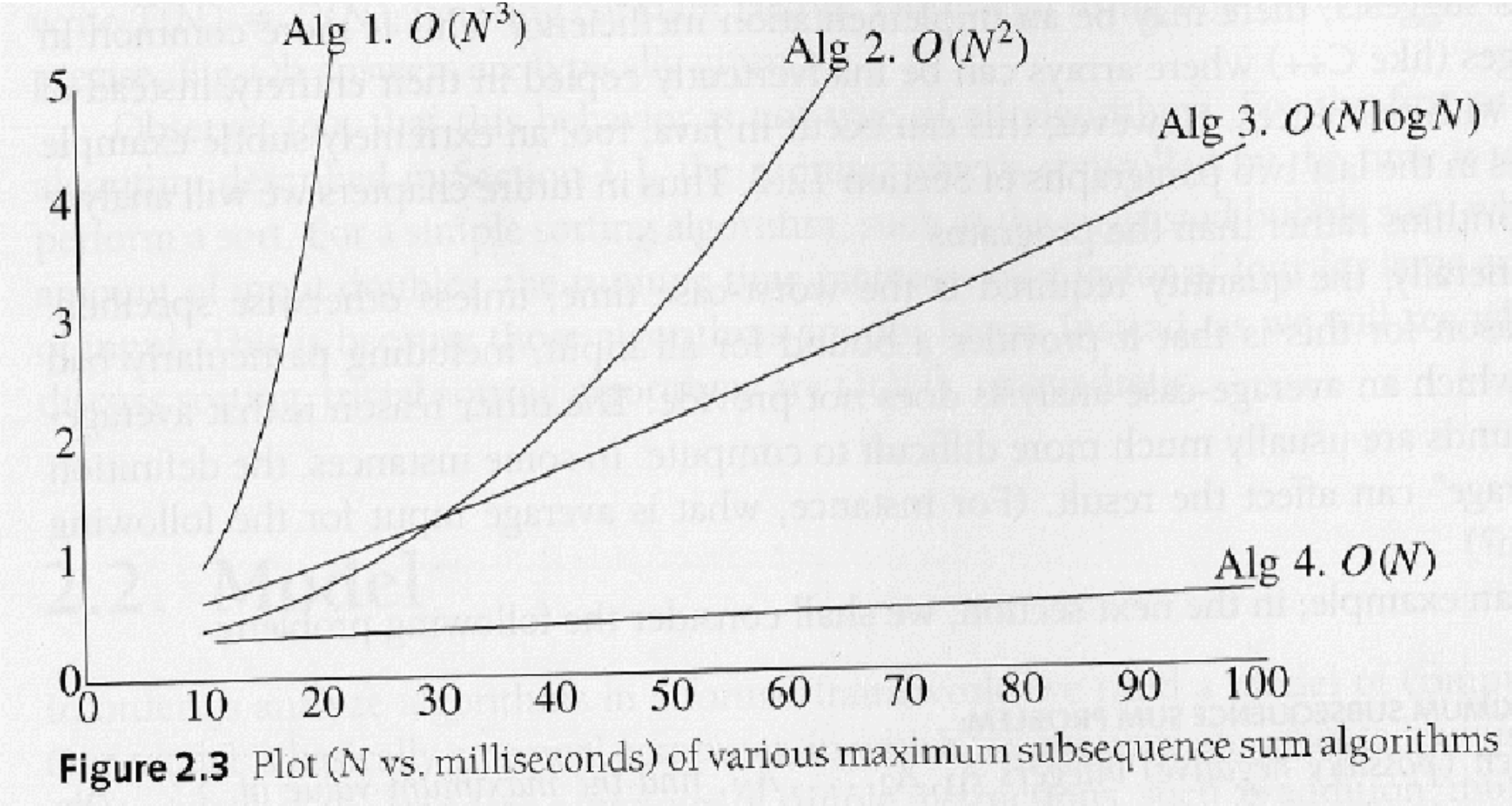

Figure 2.3 Plot (N vs. milliseconds) of various maximum subsequence sum algorithms

(valores "pequenos" de N do livro do Mark Weiss)

Subsequência de Soma Máxima

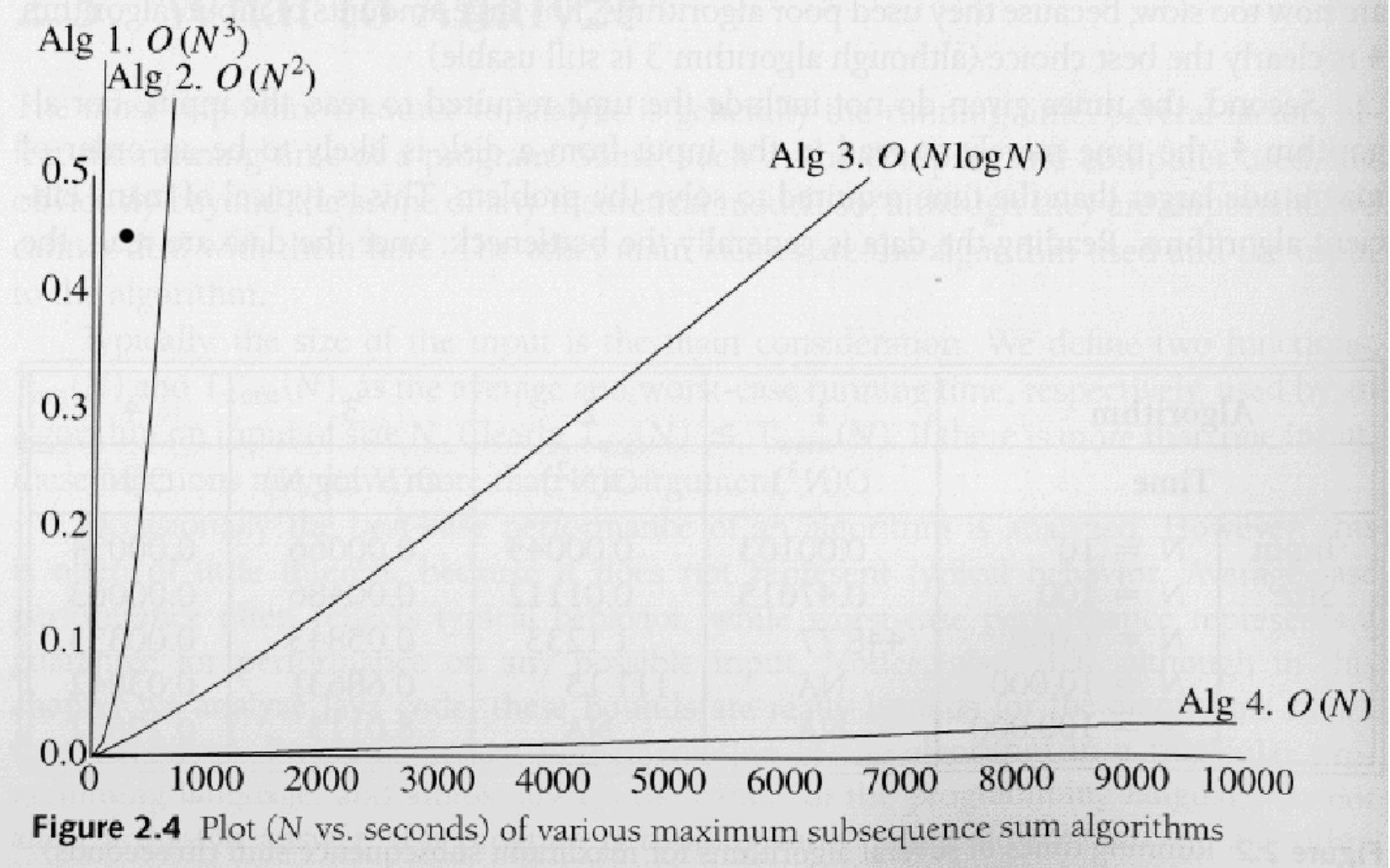

Figure 2.4 Plot (N vs. seconds) of various maximum subsequence sum algorithms

(valores "grandes" de N do livro do Mark Weiss)

Subsequência de Soma Máxima – cúbico

```
/* Algoritmo cubico para a subsequencia de soma maxima.
* seqStart e seqEnd representam a melhor subsequencia ate ao momento. */
int maxSubSum1(int a[N]){
  int maxSum = 0;

  for( int i = 0; i < N; i++ )
     for( int j = i; j < N; j++){
        int thisSum = 0;
        for( int k = i; k <= j; k++ )
           thisSum += a[k];

        if(thisSum > maxSum){
           maxSum = thisSum;
           seqStart = i;
           seqEnd = j;
        }
     }
  return maxSum;
```

Número de operações

- Operações no ciclo interior
  - inicialização k = i
  - teste k <= j
  - incremento da soma thisSum += a[k]
  - incremento k++
- Operações dominantes
  - teste e incrementos
  - inicialização

Operações de incremento da soma:
(N - tamanho do vector)

Número de ternos (i, j, k) com
$1 \leq i \leq j \leq k \leq N$
é N (N-1) (N-2) /6
logo $O(N^3)$ para o algoritmo

Ciclos exteriores têm efeito multiplicativo sobre operações no ciclo interior

O(N) operações em cada ciclo encaixado logo $O(N^3)$ para o algoritmo

Subsequência de Soma Máxima

– quadrático

```
/**
* Quadratic maximum contiguous subsequence sum algorithm.
* seqStart and seqEnd represent the actual best sequence.
*/
public static int maxSubSum2(int [] a){
  int maxSum = 0;

  for( int i = 0; i < N; i++){
     int thisSum = 0;
     for( int j = i; j < N; j++){
        thisSum += a[ j ];

        if(thisSum > maxSum){
           maxSum = thisSum;
           seqStart = i;
           seqEnd = j;
        }
     }
  }

  return maxSum;
```

Novos Algoritmos

- **Quadrático**
- Observação: soma das subsequências no ciclo interior é repetida
- Cada soma pode ser obtida a partir da anterior
  - o ciclo interior reduz-se a 1 operação - tempo constante
  - passa-se de $O(N^3)$ para $O(N^2)$

- **Linear**
- Requer melhor uso das características do problema
- Observações
  - Se $A_{i,j}$ é uma subsequência com custo negativo, $A_{i,q}$ com q>j não é a subsequência máxima
  - as subsequências contíguas da máxima têm custo negativo
  - quando se encontra uma subsequência de custo negativo pode recomeçar-se a pesquisa no elemento seguinte

Subsequência de Soma Máxima – linear

```
/**
* Linear-time maximum contiguous subsequence sum algorithm.
* seqStart and seqEnd represent the actual best sequence.
*/
public static int maxSubSum3(int [] a){
  int maxSum = 0;
  int thisSum = 0;

  for( int i = 0, j = 0; j < N; j++){
     thisSum += a[j];

     if(thisSum > maxSum){
        maxSum = thisSum;
        seqStart = i;
        seqEnd = j;
     }
     else if(thisSum < 0){
        i = j + 1;
        thisSum = 0;
     }
  }

  return maxSum;
```

Subsequência de Soma Máxima – recursivo

```
/**
* Recursive maximum contiguous subsequence sum algorithm.
* Finds maximum sum in subarray spanning a[left..right].
* Does not attempt to maintain actual best sequence.
*/
private static int maxSumRec(int [] a, int left, int right){
  int maxLeftBorderSum = 0, maxRightBorderSum = 0;
  int leftBorderSum = 0, rightBorderSum = 0;
  int center = ( left + right ) / 2;

  if(left == right) // Base case
     return a[left] > 0 ? a[left] : 0;

  int maxLeftSum = maxSumRec(a, left, center);
  int maxRightSum = maxSumRec(a, center + 1, right);
```

Subsequência de Soma Máxima – recursivo (cont.)

```
 for(int i = center; i >= left; i–) {
    leftBorderSum += a[ i ];
 if(leftBorderSum > maxLeftBorderSum)
    maxLeftBorderSum = leftBorderSum;
 }

 for(int i = center + 1; i <= right; i++){
    rightBorderSum += a[i];
 if(rightBorderSum > maxRightBorderSum)
    maxRightBorderSum = rightBorderSum;
 }

 return max3(maxLeftSum, maxRightSum,
       maxLeftBorderSum + maxRightBorderSum);
}
```

"Data Structures & Algorithm Analysis in Java", Weiss

Subsequência de Soma Máxima

– recursivo (métodos auxiliares)–

```
/* Return maximum of three integers. */
private static int max3(int a, int b, int c){
  return a > b ? a > c ? a : c : b > c ? b : c;
}
/* Driver for divide-and-conquer maximum contiguous
* subsequence sum algorithm. */
public static int maxSubSum4(int [] a){
  return maxSumRec(a, 0, N - 1);
}
```

"Data Structures & Algorithm Analysis in Java", Weiss

Subsequência de Soma Máxima

– recursivo (teste)–

```
public final class MaxSumTest{
    static private int seqStart = 0;
    static private int seqEnd = 0;

    public static void main(String [] args){
        int a[] = {4, -3, 5, -2, -1, 2, 6, -2};
        int maxSum;

        maxSum = maxSubSum1(a);
        System.out.println("Max sum is " +maxSum+ "; it goes"
          + " from " + seqStart + " to " +seqEnd);
        maxSum = maxSubSum2(a);
        System.out.println("Max sum is " +maxSum+ "; it goes"
          +" from " + seqStart + " to " + seqEnd );
        maxSum = maxSubSum3(a);
        System.out.println("Max sum is " + maxSum + "; it goes"
          +" from " + seqStart + " to " + seqEnd );
        maxSum = maxSubSum4(a);
        System.out.println("Max sum is " + maxSum);
    }
}
```

Eficiência de Estruturas de Dados

	Acesso	Comentário
Pilha	Apenas ao elemento mais recente O(1)	Muito rápido
Fila	Apenas ao elemento menos recente O(1)	Muito rápido
Lista Ligada	Qualquer item O(N)	
Árvore de Pesquisa	Qualquer item por nome ou ordem O(log N)	Caso médio; em árvores especiais é pior caso
Tabela de Dispersão	Qualquer item por nome O(1)	Quase garantido
Fila de Prioridade	Acesso ao mínimo: O(1) Apagar mínimo: O(log N)	Inserção: O(1) caso médio, O(log N) pior caso