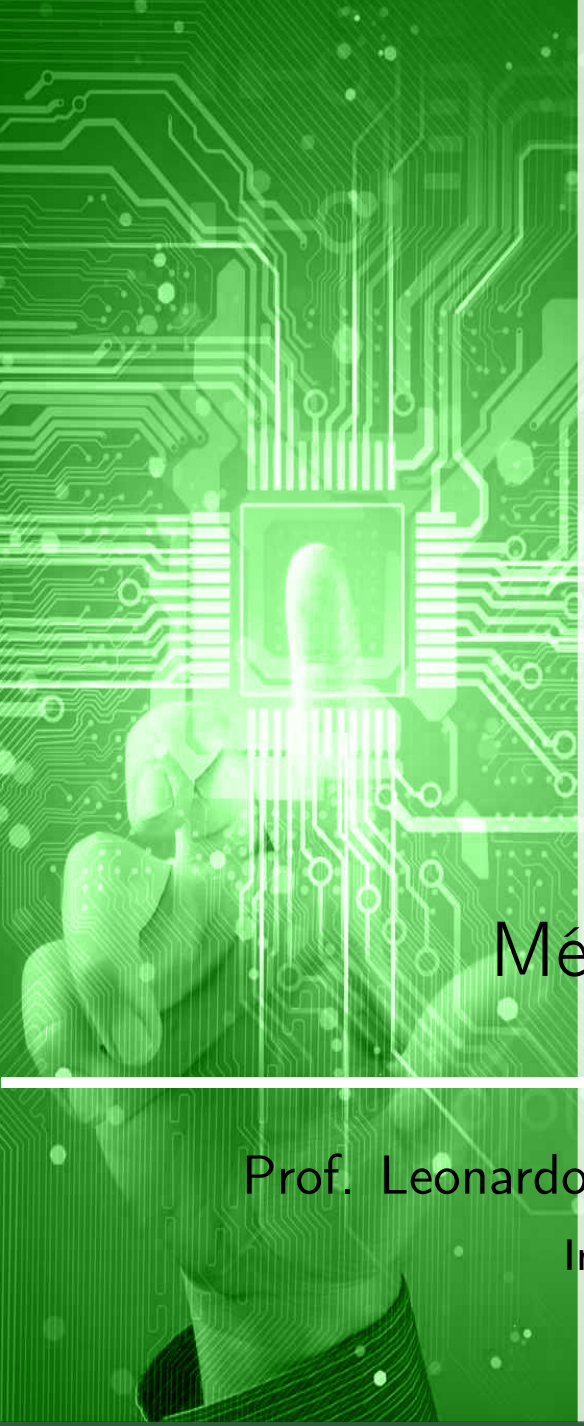


Estruturas de Dados II
Métodos de ordenação eficientes

Prof. Leonardo C. R. Soares - *leonardo.soares@ifsudestemg.edu.br*

Instituto Federal do Sudeste de Minas Gerais

22 de fevereiro de 2024

Disciplina

Estruturas de dados II

Ementa:

- Métodos de busca e ordenação;
- Teoria dos grafos;
- Introdução à análise e complexidade de algoritmos.

Disciplina

Estruturas de dados II

Ementa:

- Métodos de busca e ordenação;
- Teoria dos grafos;
- Introdução à análise e complexidade de algoritmos.

Aulas:

- Quinta-feira 10h30 - 12h00m
- Sexta-feira 10h30m - 12h00m

Estruturas de dados II

Avaliações: Serão realizadas três atividades avaliativas individuais e sem consulta. Adicionalmente, serão aplicados exercícios avaliativos valendo 1pt.

- 04/04/2024 - 3pts
- 17/05/2024 - 3pts
- 02/07/2024 - 3pts

Estruturas de dados II

Avaliações: Serão realizadas três atividades avaliativas individuais e sem consulta. Adicionalmente, serão aplicados exercícios avaliativos valendo 1pt.

- 04/04/2024 - 3pts
- 17/05/2024 - 3pts
- 02/07/2024 - 3pts

Horários de atendimento[1]**:**

- Terça-feira 16:00 - 18:00
- Sexta-feira 13:00 - 15:00

Informações detalhadas sobre a disciplina podem ser encontradas no **plano de ensino** disponível para consulta pelo SIGAA.

[1] Os atendimentos devem ser marcados com antecedência por e-mail. Em semana de provas, atendimento apenas na sexta, entre 13:00 e 17:00.

Tirando o pó dos dedos

Exercícios rápidos

- Escreva um método em Java que ordene um dado vetor.
- Escreva um método em Java que receba como parâmetro dois vetores inteiros ordenados e retorne um vetor contendo o conteúdo dos dois vetores anteriores, ordenado.

Divisão e Conquista

Introdução

- **Divisão:** Para facilitar a resolução de problemas com entradas grandes, divide-se a entrada em pedaços menores;

Divisão e Conquista

Introdução

- **Divisão:** Para facilitar a resolução de problemas com entradas grandes, divide-se a entrada em pedaços menores;
- **Conquista:** Cada subproblema (pedaço) é abordado separadamente;

Divisão e Conquista

Introdução

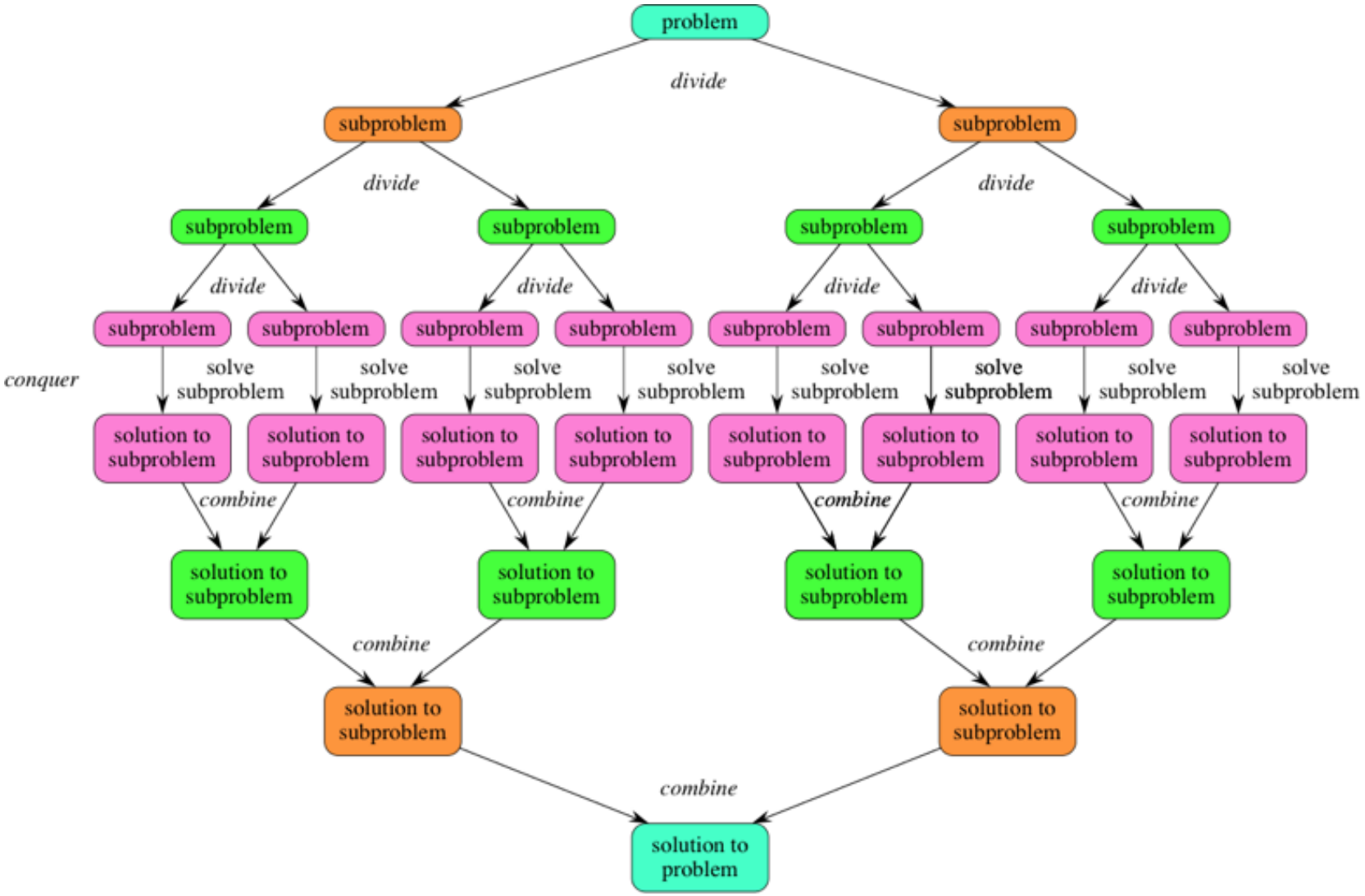
- **Divisão:** Para facilitar a resolução de problemas com entradas grandes, divide-se a entrada em pedaços menores;
- **Conquista:** Cada subproblema (pedaço) é abordado separadamente;
- Ao final, a solução dos subproblemas são combinados para gerar a solução do problema original;

Divisão e Conquista

Introdução

- **Divisão:** Para facilitar a resolução de problemas com entradas grandes, divide-se a entrada em pedaços menores;
- **Conquista:** Cada subproblema (pedaço) é abordado separadamente;
- Ao final, a solução dos subproblemas são combinados para gerar a solução do problema original;
- A técnica de divisão e conquista consiste sempre destes três passos. Dividir, conquistar e combinar.

Divisão e Conquista



Divisão e Conquista

Algoritmos de ordenação que utilizam divisão e conquista

- MergeSort: Sempre divide o problema de forma balanceada (subproblemas de mesmo tamanho);
- QuickSort: Utiliza o conceito de **pivô** para dividir o problema em subproblemas (subproblemas de tamanhos diferentes).

MergeSort

Definição

MergeSort é um algoritmo de ordenação que utiliza o padrão de projeto **divisão e conquista**. Sendo S um vetor de n elementos, seu funcionamento pode se dividido em:

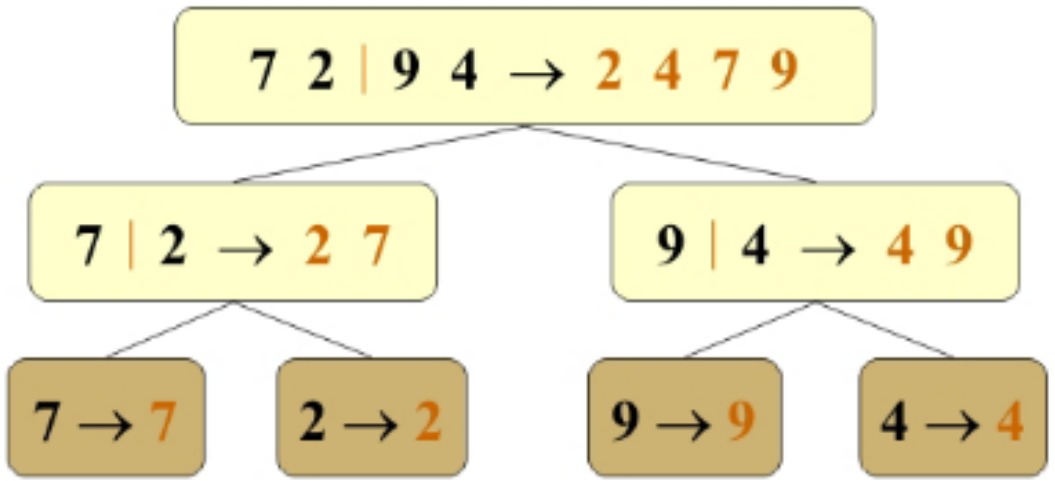
1. ***Divisão:*** se S tem zero ou um elemento, retorna-se S imediatamente, já ordenado. Em qualquer outro caso (S tem pelo menos dois elementos), removem-se todos os elementos de S e colocam-se em duas sequências, S_1 e S_2, cada uma contendo aproximadamente a metade dos elementos de S, ou seja, S_1 contém os primeiros $\lceil n / 2 \rceil$ elementos de S e S_2 contém os $\lfloor n / 2 \rfloor$ elementos restantes.
2. ***Recursão:*** ordenam-se recursivamente as sequências S_1 e S_2.
3. ***Conquista:*** os elementos são colocados de volta em S, unindo as sequências S_1 e S_2 em uma sequência ordenada. [a]

[a] Retirado do livro Estruturas de Dados e Algoritmos em Java.

MergeSort

Definição

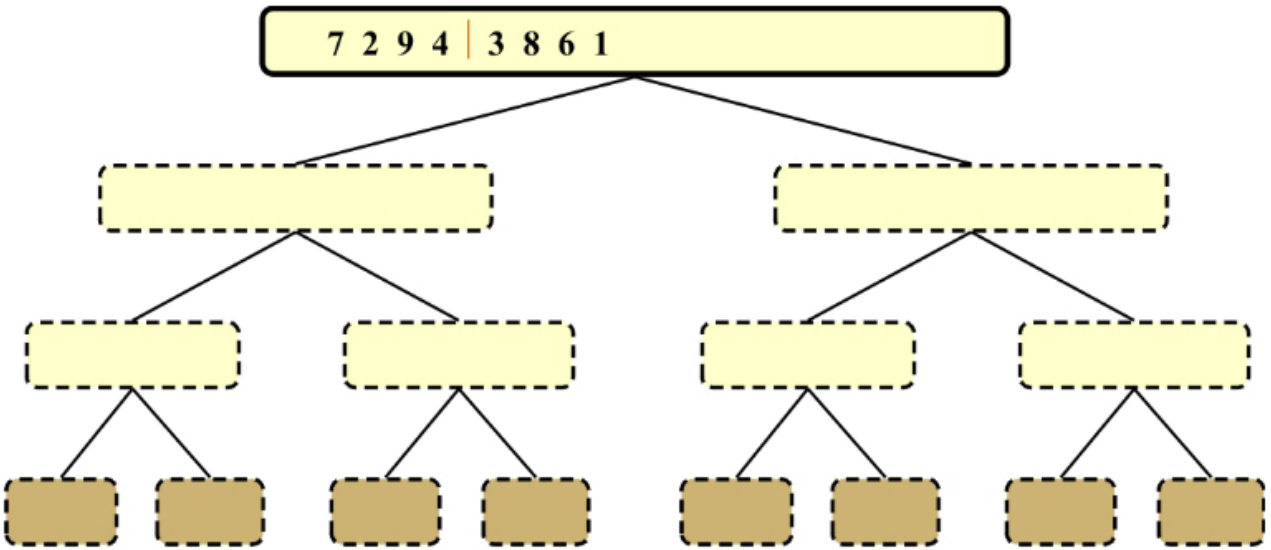
Pode-se visualizar a execução do mergesort utilizando-se uma árvore binária. Cada nodo representa uma chamada recursiva do método. Os nodos externos da árvore representam os elementos individuais do conjunto sendo ordenado, que não fazem chamadas recursivas.



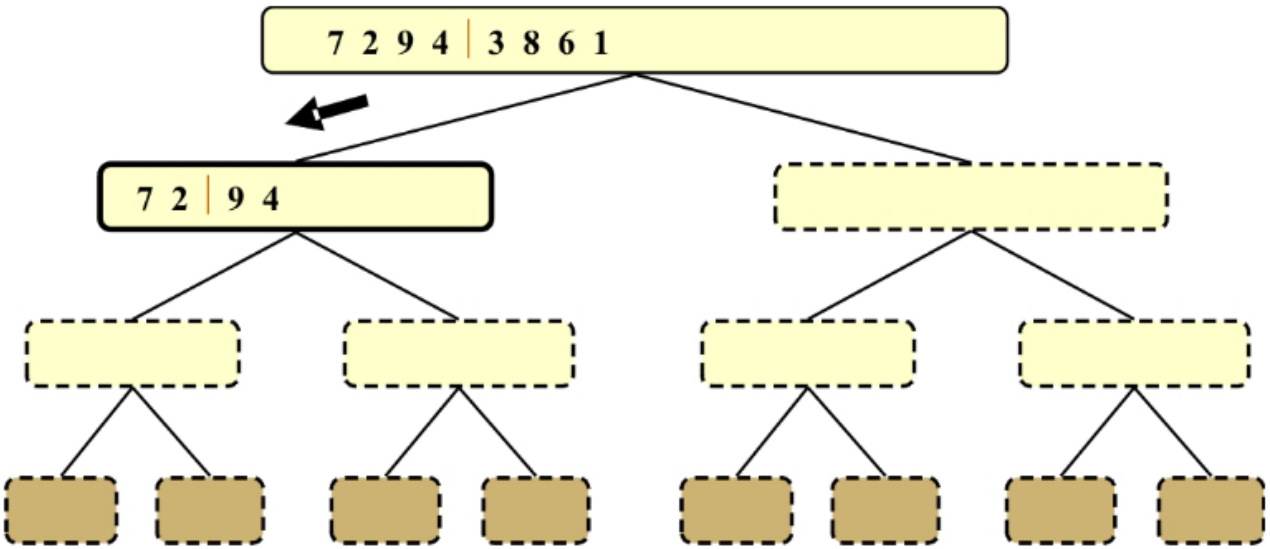
[a]

[a] Retirado das notas de aula do professor Túlio Toffolo - UFOP

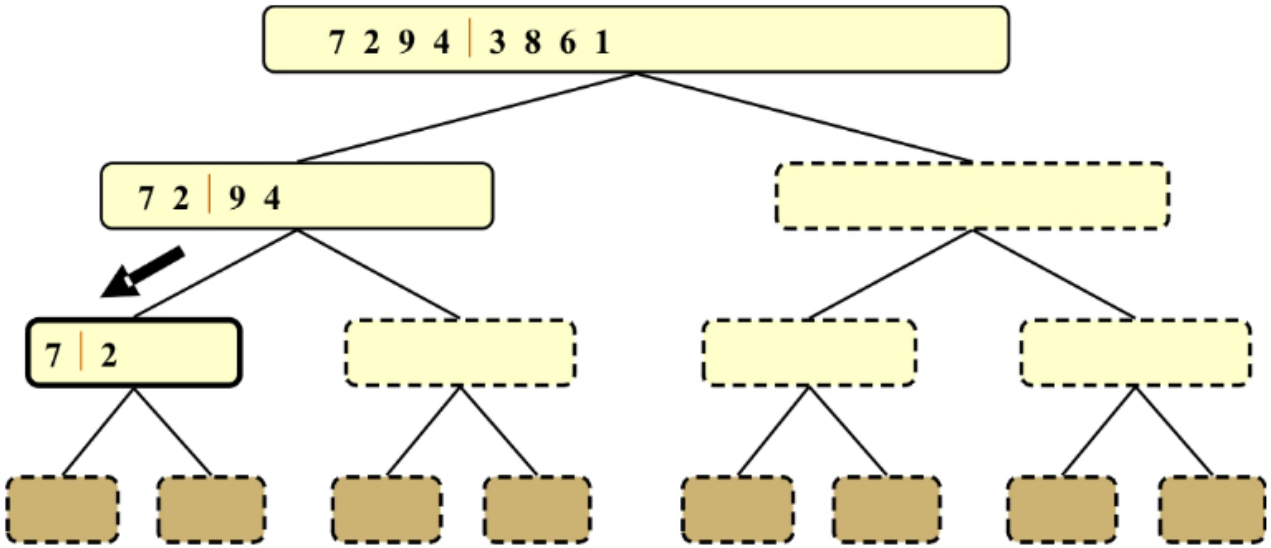
MergeSort



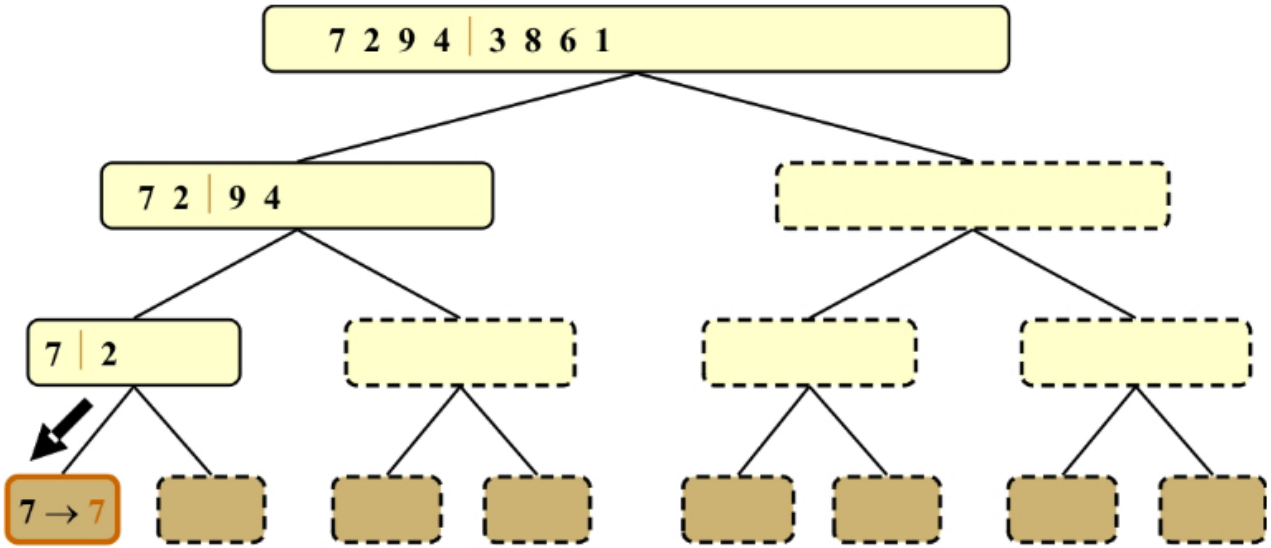
MergeSort



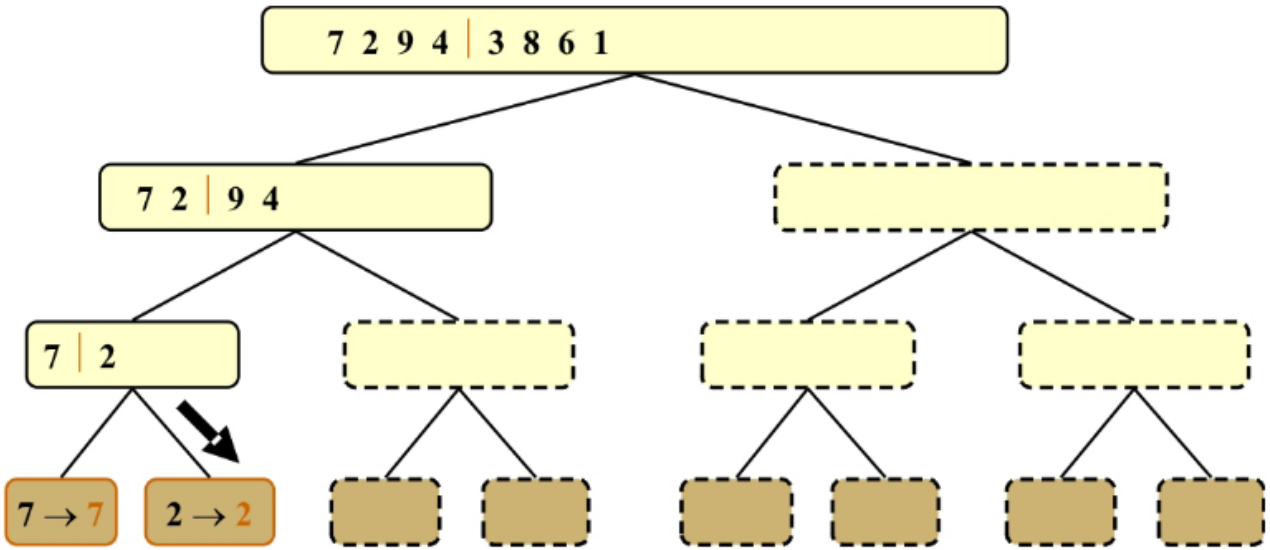
MergeSort



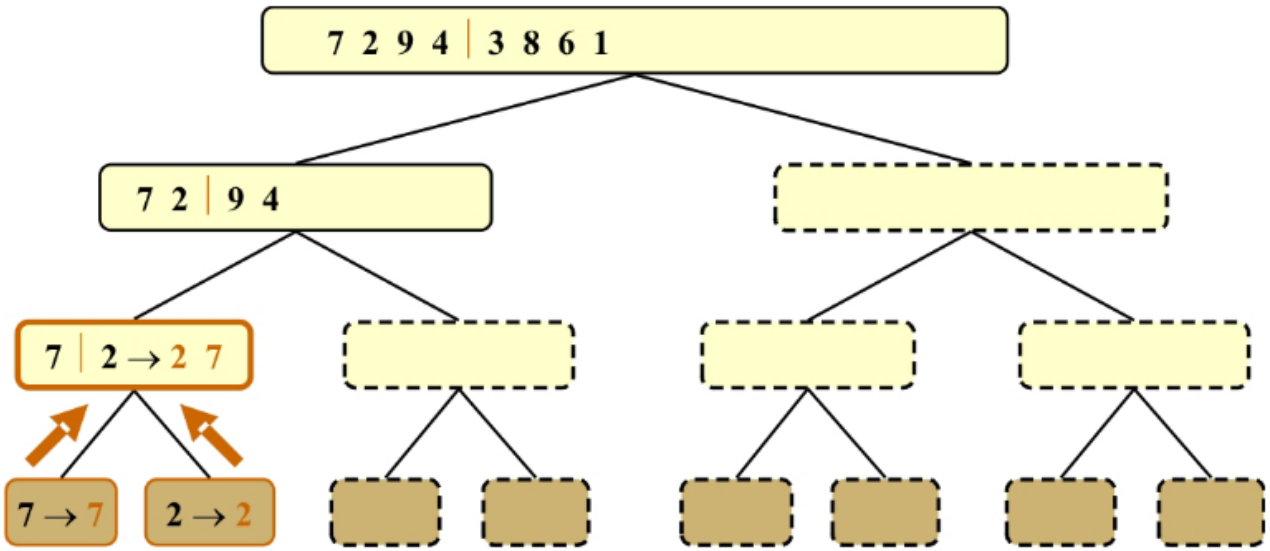
MergeSort



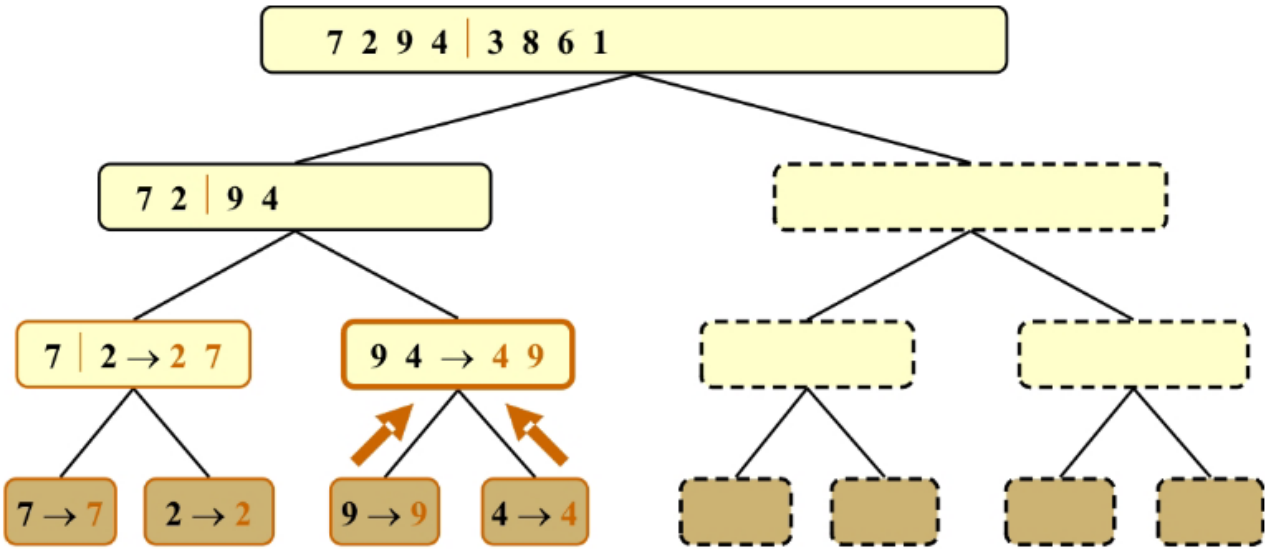
MergeSort



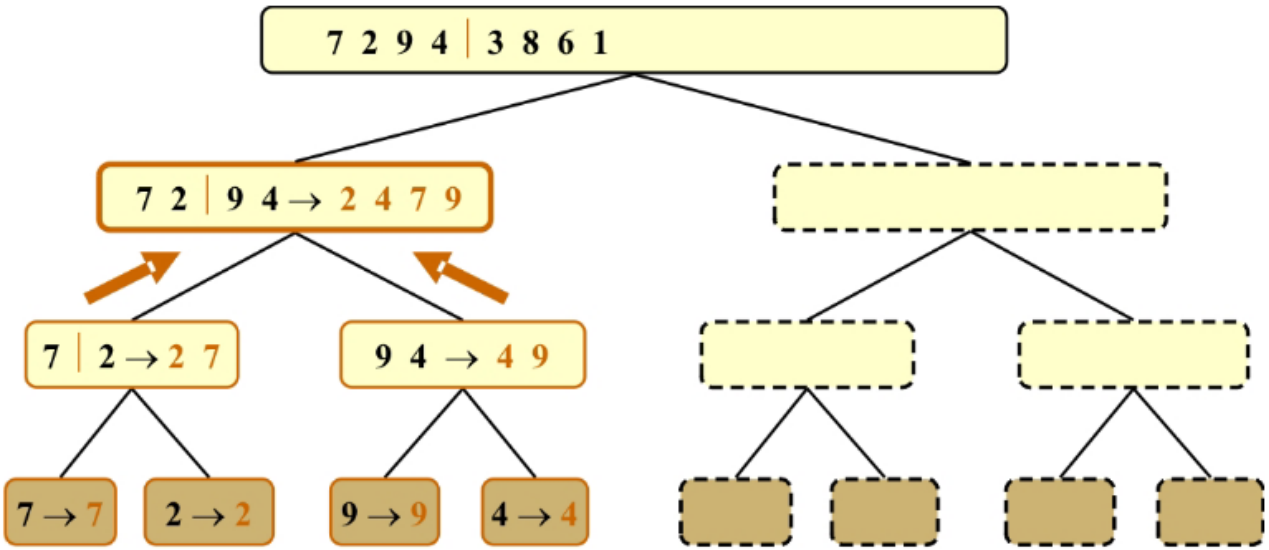
MergeSort



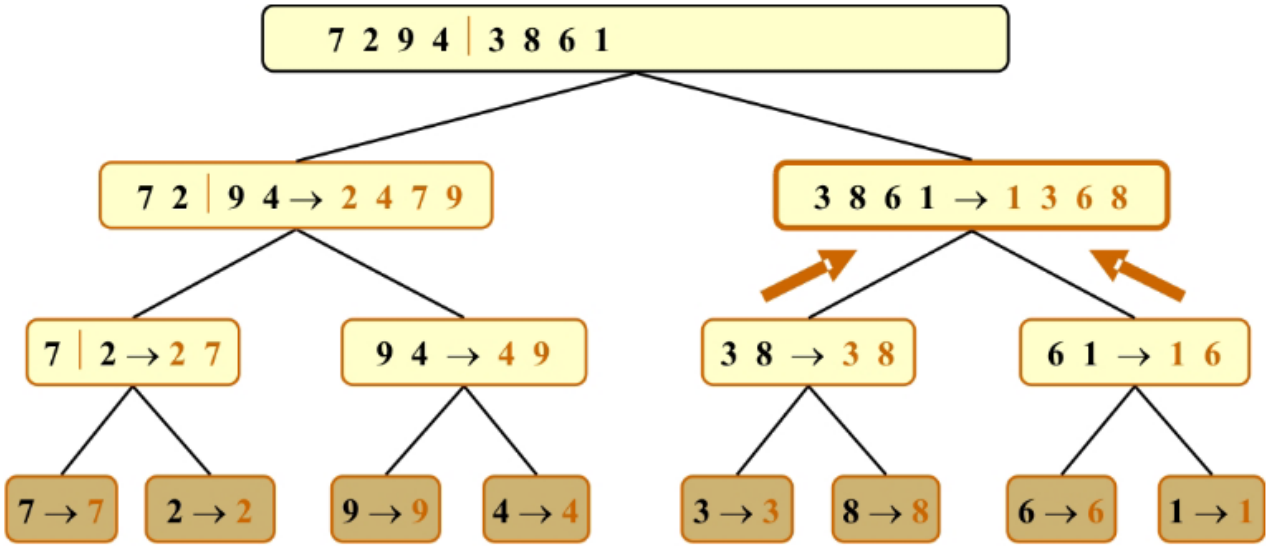
MergeSort



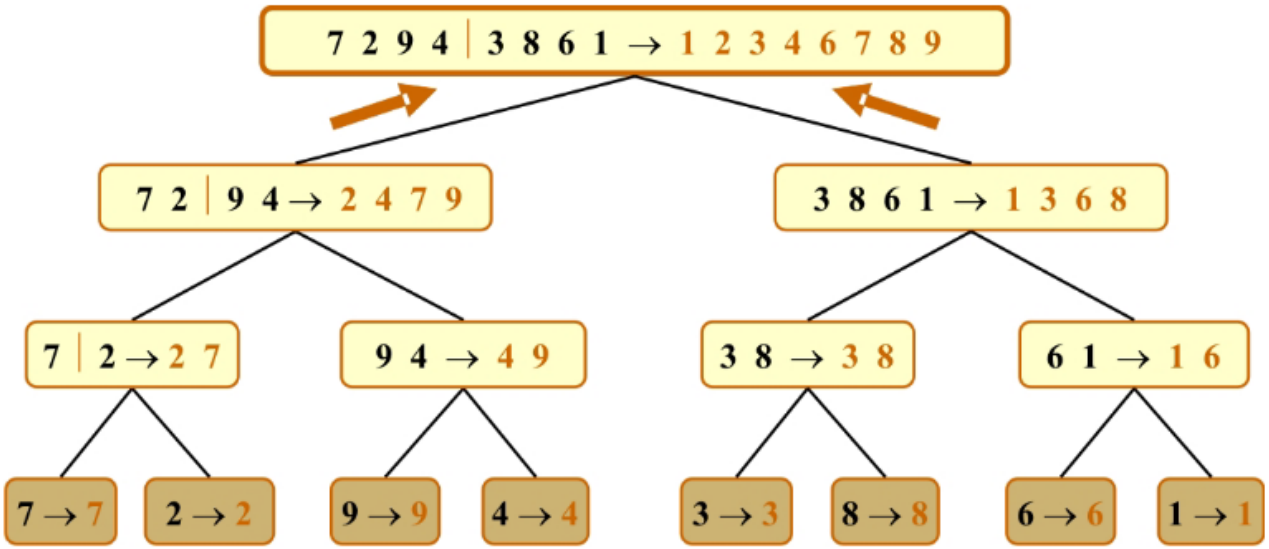
MergeSort



MergeSort


7 2 9 4 | 3 8 6 1
7 2 | 9 4 → 2 4 7 9
3 8 6 1 → 1 3 6 8
7 | 2 → 2 7
9 4 → 4 9
3 8 → 3 8
6 1 → 1 6
7 → 7
2 → 2
9 → 9
4 → 4
3 → 3
8 → 8
6 → 6
1 → 1

MergeSort


7 2 9 4 | 3 8 6 1 → 1 2 3 4 6 7 8 9
7 2 | 9 4 → 2 4 7 9
3 8 6 1 → 1 3 6 8
7 | 2 → 2 7
9 4 → 4 9
3 8 → 3 8
6 1 → 1 6
7 → 7
2 → 2
9 → 9
4 → 4
3 → 3
8 → 8
6 → 6
1 → 1

MergeSort

Análise do algoritmo

- A altura **h** da árvore de recursão é **O(log n)**;
- A quantidade de operações em cada nível da árvore é assintoticamente igual a **O(n)**;
- Logo, o algoritmo é **O(n log n)**.

MergeSort

Análise do algoritmo

- A altura **h** da árvore de recursão é **O(log n)**;
- A quantidade de operações em cada nível da árvore é assintoticamente igual a **O(n)**;
- Logo, o algoritmo é **O(n log n)**.

Importante destacar que a função merge requer um vetor auxiliar, o que aumenta o consumo de memória e apresenta-se como uma das desvantagens deste algoritmo.

MergeSort - Implementação recursiva em Java

```java
public static void mergeSort(int []a, int n){
    if (n < 2)
        return;
    int meio = n/2;
    int[] esquerda = new int[meio];
    int[] direita = new int[n - meio];
    for (int i=0;i<meio;i++)
        esquerda[i] = a[i];
    for (int i=meio; i<n; i++)
        direita[i-meio] = a[i];
    mergeSort(esquerda, meio);
    mergeSort(direita, n - meio);

    merge(a, esquerda, direita, meio, n - meio);
}
```

MergeSort - Implementação recursiva em Java

```java
public static void merge(int[] a, int[] esq, int[] dir, int nE, int nD){
    int i=0,j=0,k=0;
    while((i < nE) && (j < nD)){
        if(esq[i] <= dir[j]){
            a[k] = esq[i];
            i++;
        }else{
            a[k] = dir[j];
            j++;
        }
        k++;
    }
    while(i < nE)
        a[k++] = esq[i++];

    while(j < nD)
        a[k++] = dir[j++];
}
```

Exercícios

Dada a sequência de números, **3 4 9 2 5 1 8**, ordene em ordem crescente utilizando o algoritmo MergeSort, apresentado a sequência dos números a **cada** passo do algoritmo.

QuickSort

Definição

QuickSort é um algoritmo de ordenação **in-place** que utiliza o padrão de projeto **divisão e conquista**. Sendo S um vetor $S[p..r]$:

Divisão: Particionar (reorganizar) o arranjo $A[p \,..\, r]$ em dois subarranjos (possivelmente vazios) $A[p \,..\, q-1]$ e $A[q+1 \,..\, r]$ tais que, cada elemento de $A[p \,..\, q-1]$ é menor ou igual a $A[q]$ que, por sua vez, é menor ou igual a cada elemento de $A[q+1 \,..\, r]$. Calcular o índice q como parte desse procedimento de particionamento.

Conquista: Ordenar os dois subarranjos $A[p \,..\, q-1]$ e $A[q+1 \,..\, r]$ por chamadas recursivas a quicksort.

Combinação: Como os subarranjos já estão ordenados, não é necessário nenhum trabalho para combiná-los: o arranjo $A[p \,..\, r]$ inteiro agora está ordenado. [a]

[a] Cormen, Thomas. Algoritmos - Teoria e Prática. Disponível na Minha Biblioteca

QuickSort

O seguinte procedimento implementa o quicksort:

QUICKSORT(A, p, r)

1 **if** $p < r$

2 $\quad q =$ PARTITION(A, p, r)

3 $\quad$ QUICKSORT($A, p, q - 1$)

4 $\quad$ QUICKSORT($A, q + 1, r$)

Para ordenar um arranjo A inteiro, a chamada inicial é QUICKSORT(A, 1, A·*comprimento*).

Particionamento do arranjo

A chave para o algoritmo é o procedimento PARTITION, que reorganiza o subarranjo $A[p \,..\, r]$ no lugar.

PARTITION(A, p, r)

1 $\quad x = A[r]$

2 $\quad i = p - 1$

3 $\quad$ **for** $j = p$ **to** $r - 1$

4 $\quad\quad$ **if** $A[j] \leq x$

5 $\quad\quad\quad i = i + 1$

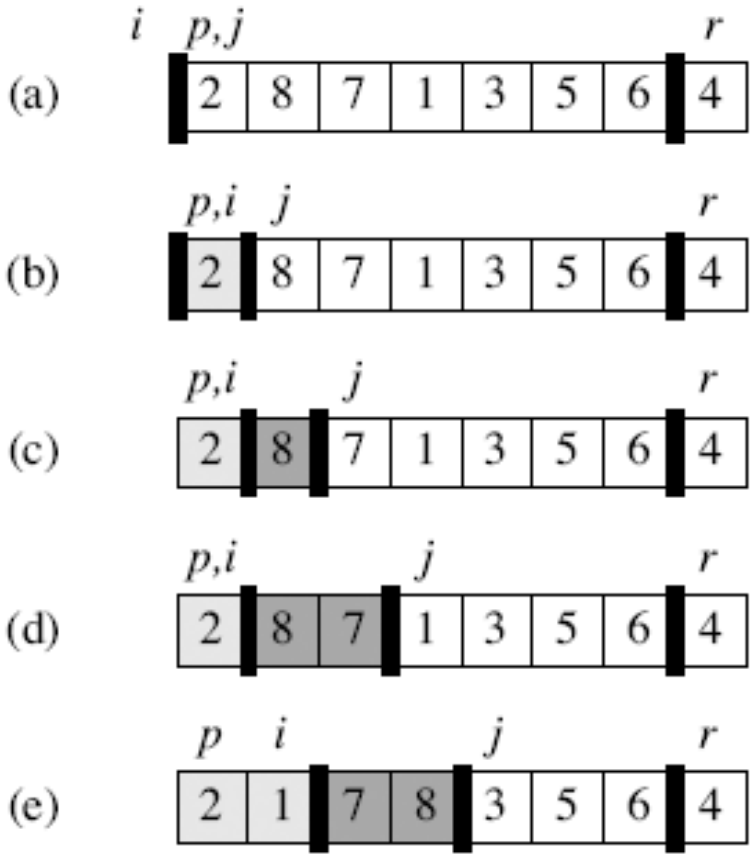
6 $\quad\quad\quad$ trocar $A[i]$ por $A[j]$

7 $\quad$ trocar $A[i + 1]$ por $A[r]$

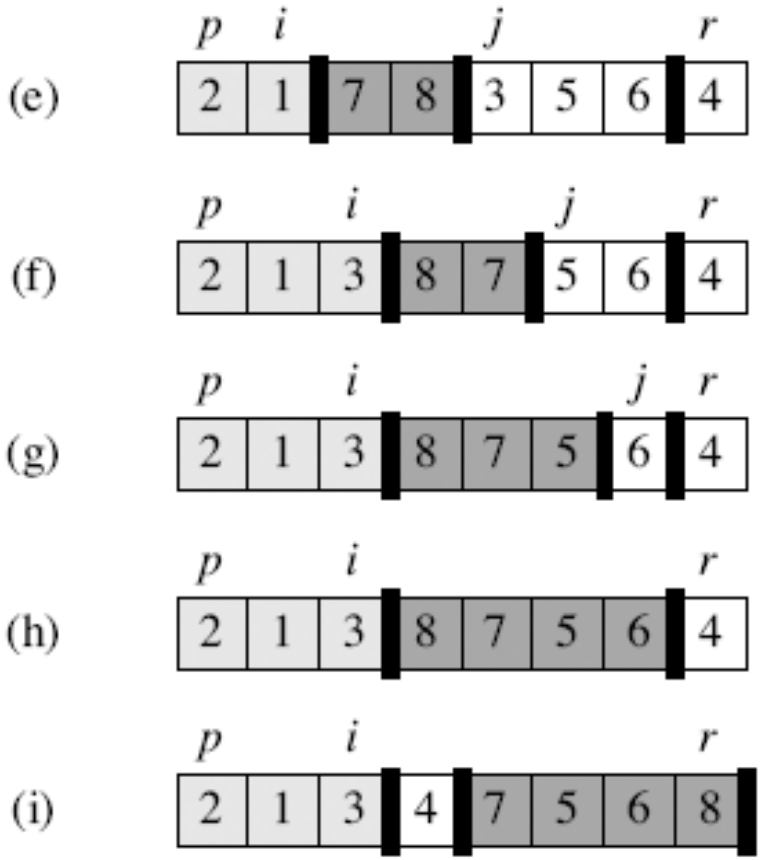
8 $\quad$ **return** $i + 1$

QuickSort

A figura abaixo mostra o funcionamento do algoritmo *Partition* para um arranjo de oito elementos.



QuickSort


p i j r
(e) 2 1 7 8 3 5 6 4
p i j r
(f) 2 1 3 8 7 5 6 4
p i j r
(g) 2 1 3 8 7 5 6 4
p i r
(h) 2 1 3 8 7 5 6 4
p i r
(i) 2 1 3 4 7 5 6 8

QuickSort

```java
public static void quickSort(int[]a, int inicio, int fim){
    if (inicio<fim){
        int q = particiona(a, inicio, fim);
        quickSort(a, inicio, q-1);
        quickSort(a, q+1, fim);
    }
}
```

QuickSort

```
public static int particiona(int[] a, int inicio, int fim){
    int pivo = a[fim];
    int i = inicio-1;
    for (int j=inicio;j<fim-1;j++){
        if (a[j]<=pivo){
            i++;
            int aux = a[i];
            a[i] = a[j];
            a[j] = aux;
        }
    }
    int aux = a[fim];
    a[fim]=a[i+1];
    a[i+1]=aux;
    return i+1;
}
```

QuickSort

Análise do algoritmo

Embora possua complexidade para o melhor caso e caso médio igual a **O (n log n)**, o pior caso para o algoritmo possui complexidade assintótica limitada por **O(nˆ2)**. O pior caso ocorre quando, sistematicamente, o pivô é escolhido como um dos extremos de um arquivo já ordenado.

Exercícios

- Ilustre todas as etapas do algoritmo apresentado para ordenação de um arranjo S=[3,9,5,2];
- Ilustre todas as etapas do algoritmo apresentado para ordenação de um arranjo S=[1,2,3,4];
- Escreva uma nova versão do algoritmo *QuickSort* em que o elemento **pivô** seja o elemento central do arranjo.


thank you
danke
gracias
merci
obrigado
teşekkür ederim
ngiyabonga
спасибо
dziękuję
bedankt
sukriya
go raibh maith agat
mochchakkeram
kop khun krap
terima kasih
arigatô
dakujem
grazie
dank je
tapadh leat
takk
sagolun
hvala
mauruuru
tack
mersi
barka
welalin
vinaka
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
mamnun
trugarez
мерси
shukriya
dhanyavadagalu
diolch
xiexie
tanemirt
rahmet
taiku
sulpay
gracies
chnorakaloutioun
najis tuke
kam sah hamnida
rahmat
didi madloba
mèsi
sobodi
dėkuji
enkosi
bayarlalaa
gracie
dhanyavad
kiitos
dankie
nanni
nandri
taafetai lava
Баярлалаа
рахмат
謝謝
spas
kia ora
dankon
akun
köszönöm
djiere dieuf
дякую
ευχαριστώ
ありがとう
감사합니다