University of Zurich
Department of Informatics
Database Technology Group
Prof. Dr. Anton Dignös

Informatics II
Exercise 4

March 15, 2021

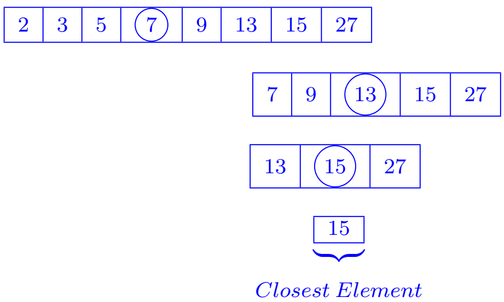
Goal:

- Draw a tree that illustrates the divide and conquer approach.
- Practise divide-and-conquer algorithms.
- Draw a recursion tree, and estimate the asymptotic upper bound
- Calculate the symptotic tight bound of recurrences.
- Practise Master Theorem.

Divide and Conquer

Task 1. Given an array $A[...]$ of n integers sorted in ascending order and a target integer t, write a divide-and-conquer algorithm that finds the closest number to t in the array A. One integer a is closer to t than another integer b if $|a - t| < |b - t|$.

a) Draw a tree to illustrate the process of finding the closest number to $t = 20$ in the array $A =$ [2,3,5,7,9,13,15,27] according to your divide-and-conquer algorithm.



b) Implement your divide-and-conquer algorithm that takes an array A, n, and a target integer t as input, and returns the closest number to t in the array A. Use C code for your solution.

```c
// Find the closest number in a sorted array
#include <stdio.h>
#include <stdlib.h>

int find_closer_in_two_values(int a, int b, int t) {
  if (abs(a - t) < abs(b - t)) {
    return a;
  }
```

```
  return b;
}

int find_closest(int A[], int n, int t) {
  if (t ≤A[0]) {
    return A[0];
  }
  if (t ≥A[n - 1]) {
    return A[n - 1];
  }
  // Now perform binary search
  int i = 0;
  int j = n;
  int mid = 0;
  while (i < j) {
    mid = (i + j) / 2;
    if (t == A[mid]) {
      return A[mid];
    }
    if (t < A[mid]) {
      // In this case, target is smaller than A[mid], all elements
      //after index mid are exclude. Binary search paradigm is applied.

      if (mid > 0 && t > A[mid - 1]) {
        // when A[mid - 1] > t, we can determine that the closest number to
        // t is either A[mid] or A[mid - 1]
        return find_closer_in_two_values(A[mid - 1], A[mid], t);
      }

      // we can't determine the closest number, so we continue to our search
      // between indice i and j = mid.
      j = mid;
    } else {
      // In this case, target is larger than x[mid], all elements
      //before index mid are exclude. Binary search paradigm is applied.

      if (mid < n - 1 && t < A[mid + 1]) {
        // when A[mid + 1] < t, we can determine that the closest number to
        // t is either A[mid] or A[mid + 1]
        return find_closer_in_two_values(A[mid], A[mid + 1], t);
      }

      // we can't determine the closest number, so we continue to our search
      // between indice i = mid + 1 and j
      i = mid + 1;
    }
  }
  // after the search, there is only one element left
  return A[mid];
}

int main() {
  int A[100];
  int i = 0;
  int t = 0;
  int n = 0;
```

```
  printf("Values of the array separated by spaces (non-number to stop): ");

  while (scanf("%d", &A[i]) == 1) {
    i++;
  }
  n = i;
  scanf("%*s");
  printf("Target t: ");
  scanf("%d", &t);

  printf("Result: ");

  printf("%d\n", find_closest(A, n, t));
  return 0;
}

// Linux, Mac: gcc task04_1.c -o task04_1;
// ./task04_1
```

Task 2. Given an array $A[...]$ of n integers, write an algorithm to calculate the number of inversions in the array A. For array A, an *inversion* is a pair of positions (i, j) where $1 \leq i < j \leq n$ and $A[i] > A[j]$. Assume $A = \{3, 2, 1\}$, there are three inversions in A – $(1, 2), (1, 3)$ and $(2, 3)$. For example, $(1, 2)$ is an inversion because $A[1] > A[2]$.

a) Implement a solution with $O(n^2)$ time complexity in C.

```
int naive_inversion_count(int A[], int n) {
  int inversion_count = 0;
  for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
      if (A[i] > A[j]) {
        inversion_count = inversion_count + 1;
      }
    }
  }
  return inversion_count;
}
```

How to call this function will be illustrated in the next subtask.

b) Implement a divide-and-conquer solution in C. *Hint: think about merge sort, can you slightly modify and apply it here?*

```
// in merge, A[low,...,mid] are sorted and A[mid + 1,...,high] are sorted
//calculate number of inversion
int merge(int A[], int tmp[], int low, int mid, int high) {
  int i = low;
  int j = mid + 1;
  int inversionCount = 0;
  int k = low;
  while (i <=mid && j <=high) {
    if (A[i] > A[j]) {
      // find inversions: A[i], A[i + 1]... A[mid] are all bigger than A[j]
      // i < j, i + 1 < j and mid < j, so increase inversionCount by (mid + 1 - i )
      inversionCount = inversionCount + (mid + 1 - i);
      tmp[k] = A[j];
```

```c
            k++;
            j = j+1;
        } else {
            tmp[k] = A[i];
            k++;
            i = i+1;
        }
    }
    // moving remaining elements in Array A with i <=index <=mid to Array tmp
    while(i <=mid) {
        tmp[k] = A[i];
        k++;
        i++;
    }
    // moving remaining elements in Array A with j <=index <=high to Array tmp
    while(j <=high) {
        tmp[k] = A[j];
        k++;
        j++;
    }

    //move sorted elements from Array tmp to Array A
    for (i = low; i <=high; i++) {
        A[i] = tmp[i];
    }
    return inversionCount;
}

int mergeSort(int A[], int tmp[], int low, int high) {
    // base case, there is one single integer.
    if (low >=high) {
        return 0;
    }
    int mid = (low + high) / 2;
    int inversionCount = 0;
    //sort A[low,...,mid] and compute the number of inversion if A[low, ..., mid]
    inversionCount = inversionCount + mergeSort(A, tmp, low, mid);
    //sort A[mid + 1,..., high] and compute the number of inversion if A[mid + 1, ..., high]
    inversionCount = inversionCount + mergeSort(A, tmp, mid + 1, high);
    // calculate number of inversion in A[low, ..., high].
    inversionCount = inversionCount + merge(A, tmp, low, mid, high);
    return inversionCount;
}

int main() {
    int A[100];
    int tmp[100];
    int n = 0;
    printf("Values of the array separated by spaces (non-number to stop): ");
    while (scanf("%d", &A[n]) == 1) {
        n++;
    }
    scanf("%*s");
    printf("[Naive] Number of Inversions: %d\n", naive_inversion_count(A, n));
    printf("[Divide-Conquer] Number of Inversions: %d\n",
           mergeSort(A, tmp, 0, n - 1));
```

```
}
```

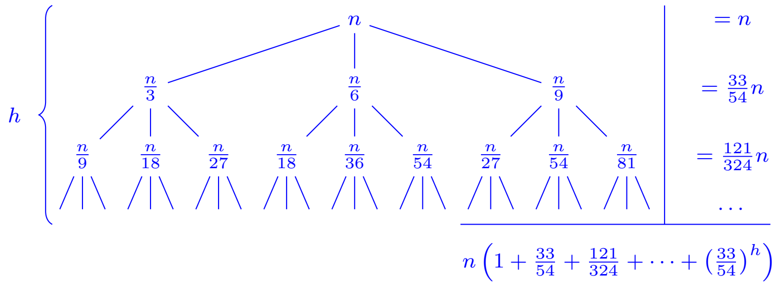
c) Calculate the asymptotic tight bound in b).
Similar to merge sort, we have:
Recurrence: T(n) = 2T(n/2) + θ(n)
Asymptotic Complexity : $\theta(n \log n)$

Recurrences

Task 3. Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{, if } n = 1 \\ T(n/3) + T(n/6) + T(n/9) + n & \text{, if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Demonstrate the tree-based computations that led to your estimate.



- The tree grows until $\frac{n}{3^h} = 1; \implies h = \log_3 n$

Guess: $O(n)$

b) Use the substitution method to prove that your estimate in (a) is correct.

Inductive Step

- $T(n) \leq cn \implies T(n/3) + T(n/6) + T(n/9) + n \leq c\frac{n}{3} + c\frac{n}{6} + c\frac{n}{9} + n$
- We want to solve $c\frac{n}{3} + c\frac{n}{6} + c\frac{n}{9} + n \leq cn$

Solution

- $c\frac{n}{3} + c\frac{n}{6} + c\frac{n}{9} + n \leq cn$
- dividing both parts by n we get $c\frac{1}{3} + c\frac{1}{6} + c\frac{1}{9} + 1 \leq c$
- $\frac{21}{54}c \geq 1 \implies c \geq \frac{54}{21}$

Asymptotic Complexity

$T(n) = O(n)$, for $c \geq \frac{54}{21}$

Task 4. Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and the case (1-3).

1. $T(n) = 4T(\frac{n}{16}) + 16\sqrt{n}$

   $a = 4$, $b = 16$, $f(n) = 16\sqrt{n}$, Case 2 (because $\frac{1}{2} = \log_{16} 4$)
   $T(n) = \Theta(\sqrt{n}\log_{16} n)$

2. $T(n) = T(\sqrt{n}) + \log n$

$$
\begin{aligned}
T(n) &= \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + \dots \\
&= \log n + \frac{1}{2}\log n + \frac{1}{4}\log n + \dots \\
&= \left(\sum_{k=0}^{\infty} (\frac{1}{2})^k\right) \log n \\
&= 2\log n
\end{aligned}
$$

   $\Rightarrow T(n) \in \Theta(\log n)$

3. $T(n) = 16T(\frac{n}{8}) + n^3$

   $a = 16$, $b = 8$, $f(n) = n^3$, Case 3:
   $T(n) \in \Theta(n^3)$

4. $T(n) = 3T(n-2) + n$

$$
\begin{aligned}
T(n) &= 3T(n-2) + n \\
&= 3(3T(n-2) + (n-2)) + n \\
&= 9T(n-4) + 4n - 6 \\
&= 9(3T(n-6) + (n-4)) + 4n - 6 = 27T(n-6) + 13n - 42 \\
&= \dots \\
\Rightarrow T(n) &= 3^k T(n-2k) + (\frac{3^k - 1}{2})n - \sum_{i=0}^{k-1} 3^i \cdot 2i
\end{aligned}
$$

   k grows until it reaches $k = \lfloor \frac{n}{2} \rfloor$, thus we have:

$$
T(n) = \frac{(3^{\lfloor \frac{n}{2} \rfloor} - 1)}{2} n - \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} 3^i \cdot 2i \leq \frac{(3^{\lfloor \frac{n}{2} \rfloor} - 1)}{2} n - 3^{\lfloor \frac{n}{2} \rfloor - 1} 2 \cdot (\lfloor \frac{n}{2} \rfloor - 1) \in \Theta(n\sqrt{3^n})
$$

5. $T(n) = \log n + T(\sqrt[3]{n})$

$$
\begin{aligned}
T(n) &= \log n + \log \sqrt[3]{n} + \log \sqrt[3]{\sqrt[3]{n}} + \dots \\
&= \log n + \frac{1}{3}\log n + \frac{1}{9}\log n + \dots \\
&= \left(\sum_{k=0}^{\infty} (\frac{1}{3})^k\right) \log n \\
&= \frac{5}{2}\log n
\end{aligned}
$$

   $\Rightarrow T(n) \in \Theta(\log n)$