DSA – Performance Lab

Sort: **bubble, insertion, quicksort**

Search: **sequence: linear, binary**

Sorting & searching

- Bubble sort **$O(n^2)$**
- Insertion sort **$O(n^2)$**
- Quicksort **O(n log n)** **worst case $O(n^2)$**

- Linear search in a sequence **O(n)**
- Binary search in a sequence **O(log n)**

What to measure

- Sort/Search Data - array of int
- Size: 1024, 2046, 4096, 8192, 16384
- Case: best, random, worst

  how do you decide these?

  what are the big-O for these?

  do some research!

Timing

- Start time
- Run test
- Stop time
- Time = Stop time – Start time

- Look at C clock and timing possibilities

Timing – other factors

- Run each test (say) 10 times
- Sum and take the average value
  - This may give you more stable results

==========

- In performance tests, the program start-up overhead for first 1 or 2 runs (here 1024, 2048) may be significant.

  I.e. the system has not reached steady state.
  - The results for the first runs are often ignored!

Linear search for x

- Best case x is the first element
- Worst case x is the last element
  what is an alternative?
- Random case what does this mean?

- Expected result O(1), O(n), O(n)
  - O(1) is constant time

Goals for this project

- To
  - Introduce performance measurement
    - Theory versus practice
  - Show the given big-O's for each algorithm and each case, are true
  - Compare algorithms
    - E.g. it is said that insertion sort is better than bubble sort **(bubble sort is a worst case sort example)**

Analysis

- Choose 3 big-O's
  1. The expected big-O e.g. $O(n^2)$
  2. A lower big-O e.g. $O(n)$
  3. A higher big-O e.g. $O(n^3)$
- The values for the expected big-O should converge to a constant
  - **See http://www.cs.kau.se/cs/education/courses/dvgb03/lab_info/index.php?Weiss=1**

Documentation

- See http://www.cs.kau.se/cs/DFR/index.php?labreqs=1
- Otherwise the format is open
- This is an “engineering project”
- You decide on the methods
- We are looking for initiative and originality!
- Good luck!

Design Framework

- See http://www.cs.kau.se/cs/education/courses/dvgb03/lab_info/index.php?PerfLabDesign=1
- Model: UI (menu) + FE + BE
- Menu: 15 cases + run all + show menu
- Focus on **1 case** e.g. **bubble sort best**
- Implement this case:
  - Menu choice **c** ➔ **FE function** ➔ **BE function**
  - **FE function** calls **BE function** **5 times** **(1024, 2048, 4096, 8192, 16384)**
  - **BE function**: (1) initialise array; (2) start timer; (3) **bubble sort**; (4) stop timer; (5) return time (stop time – start time)

Design Framework

- Reflect on what is required in the **FE** & **BE**
- **FE**
  - Call **BE** function **5 times**
  - **Each call returns a time**
  - The **5 times** are then displayed and analysed (**table**)
- **BE**
  - Requires an **array** + initialising functions
  - Requires a **timing mechanism** (start & stop)
  - Each sort / search may be executed (say) 10 times and an average result (**time**) returned ((sum of times)/10)

Table Structure (FE output)

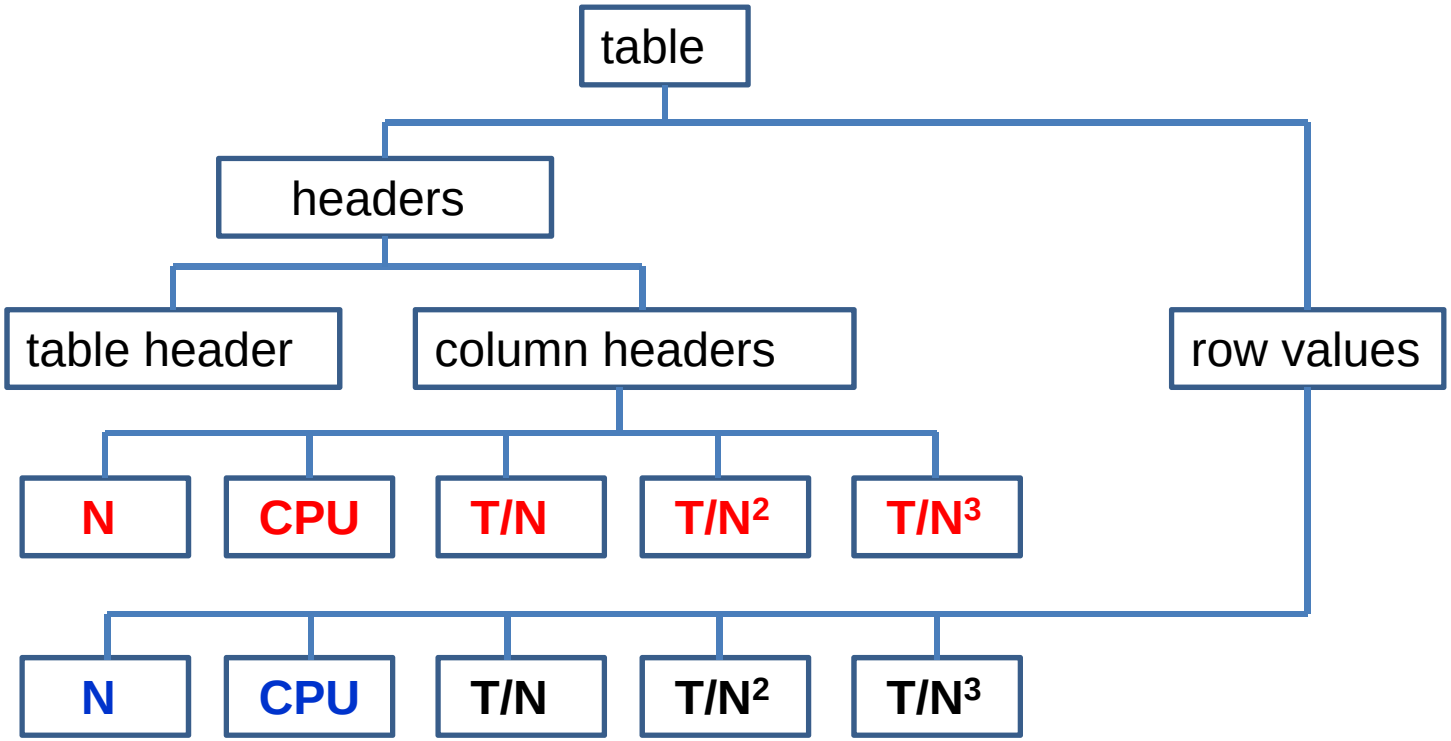
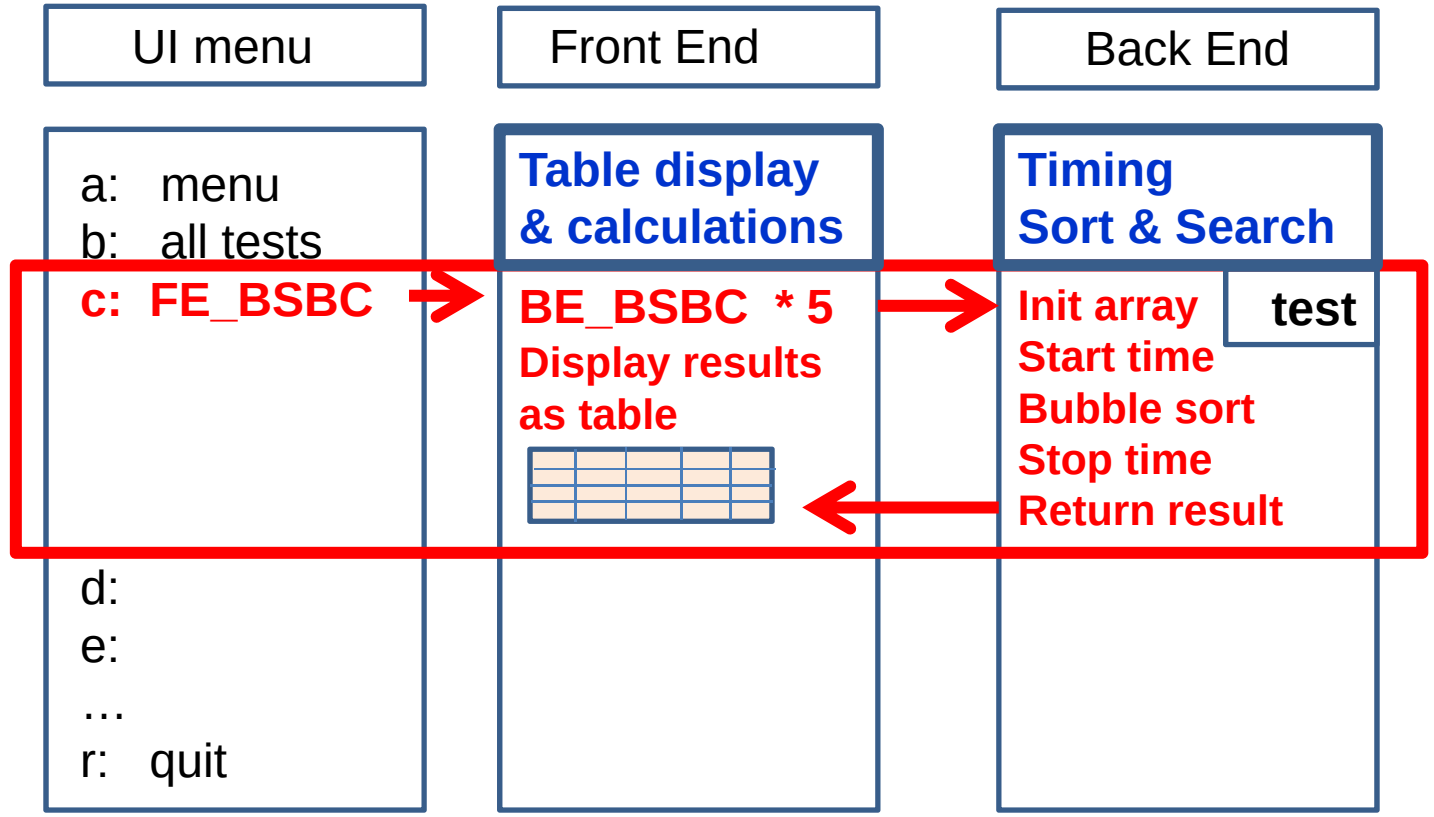


Table Display

- Note that the table display (for each table type) needs only **1 parameter**
  - **The heading for the table** string constant
- The **table types** are determined by the **analysis**
- The **column headers** may be defined globally
- The **# of rows** is controlled by the **# of tests** (5)
- The **array of timing result values** comes from the tests and may be defined globally
- All other values are **known** (1024…) or **calculated**

Visualising the Design



Generalising the Design

- Look for “magic numbers” and “parameterise” them
- **FE: 5 values:** 1024, 2048, 4096, 8192, 16384
  - **int numtest = 5** (default) but user changeable via menu
  - **int startsize = 1024** – double this for each run (numtest)
- **BE: 10 runs:**
  - **int nruns = 10** (default) but user changeable via menu
- **Table Structure – e.g.** (input is title + results array)
  - **Table type 1:** **T/N** **T/N^2** **T/N^3** **bubble / insertion sort**
  - **Table type 2:** **T/logN** **T/N** **T/N^2** **linear / binary search**
  - **etc.**
- **Further generalisation might lead to 2 FE functions and 1 BE function!**

Advantages of the Design

- By implementing **one case only**, the whole process is covered
  - **Menu choice ➔ FE call ➔ BE calls (5) ➔ results ➔ display**
  - **BE test: initialise array ➔ start timer ➔ bubble sort ➔ stop timer ➔ return result**
- Array initialisation, timing, sorting is done in the BE
- Once this case is working, the rest is copy & paste!
- The main tasks are
  - **Implementing a working timing mechanism**
  - **Implementing sort / search algorithms (total 5)**
- **A working demonstration is achieved more quickly!!!**