CUDA 프로그래밍

CUDA Programming

biztripcru@gmail.com

시간 측정

Elapsed Time

내용 contents

- Elapsed Time Calculation
- C++ Chrono features
- clock( ) function
- sleep( ) function
- argc, argv 처리

Motivation

- **We have two versions of programs**
  - CPU-based sequential execution
  - CUDA-based parallel execution
- **How much faster?**
  - time measurements are the answer !
- **Method?**
  - set the start time = current time
  - do some thing
  - set the end time = current time
  - **elapsed time = (end time – start time)**
- **compare the elapsed times !**

Wall-clock time vs CPU time

- **wall-clock time**
  - 또는 elapsed real time
  - 컴퓨터 프로그램이 실행되면서 실제로 흘러간 시간
  - 이론상 벽시계 wall-clock 로 측정해도 같은 결과
- **CPU time (CUDA 에서는 GPU time)**
  - 컴퓨터 프로그램이 실행 중에 CPU를 사용한 시간
  - system time, kernel time – 운영체제 O/S 영역에서 사용한 시간
  - user time, user CPU time – 사용자 user 영역에서 사용한 시간
- **일반적으로, CPU time < wall-clock time**
  - 다만, 병렬프로그래밍에서는 CPU time > wall-clock time 가능

C++ Standard

- **history**

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11, C++0x
2014	ISO/IEC 14882:2014	C++14, C++1y
2017	ISO/IEC 14882:2017	C++17, C++1z
2020	ISO/IEC 14882:2020	C++20, C++2a

- **modern C++ means: C++11 and later**

C++11

- **the new standard of the C++ language**
  - also known as C++0x
  - published in late 2011.

- **GNU C++ compiler requires the command line parameter –std=c++11 to compile C++11 code.**
  - **–std=c++0x** also works

- **Microsoft Visual Studio 2015 (v14) and later have complete support for C++11 features.**

Chrono data types and library

- chronograph = (스톱워치 기능이 있는) (클래식) 시계

- C++ 11 standard
  - We need a **system-independent** time measuring method...
  - with more high precision

public domain
https://en.wikipedia.org/wiki/Chronograph#/media/File:Gallet_multichron_astronomic_450x600.jpg

- **#include <chrono>**
  - using namespace std::chrono
  - provide the nano-second precision
  - **for wall-clock time !**

Duration Units

- in <chrono> header file:

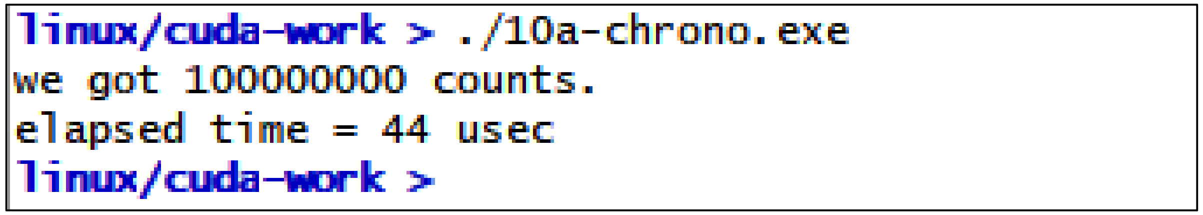
- typedef duration<int, ratio<3600> > hours;
- typedef duration<int, ratio<60> > minutes;
- typedef duration<long long> seconds;
- typedef duration<long long, milli> milliseconds; // (1 / 1,000) sec
- typedef duration<long long, micro> microseconds; // (1 / 1,000,000) sec
- typedef duration<long long, nano> nanoseconds; // (1 / 1,000,000,000) sec

Example: chrono.cpp

```
#include <stdio.h>
#include <time.h>
#include <chrono>
using namespace std;
using namespace std::chrono;

// dummy big job
void bigJob(void) {
   int count = 0;
   for (int i = 0; i < 10000; ++i) {
      for (int j = 0; j < 10000; ++j) {
         count++;
      }
   }
   printf("we got %d counts.\n", count);
}
```

Example: chrono.cpp 계속

```
int main(void) {
   system_clock::time_point  chrono_begin = system_clock::now();
   // work
   bigJob();
   // work done
   system_clock::time_point  chrono_end = system_clock::now();
   // calculation
   microseconds   elapsed_usec = duration_cast<microseconds>(chrono_end - chrono_begin);
   printf("elapsed time = %ld usec\n", (long)elapsed_usec.count());
   // done
   return 0;
}
```



```
linux/cuda-work > ./10a-chrono.exe
we got 100000000 counts.
elapsed time = 44 usec
linux/cuda-work >
```

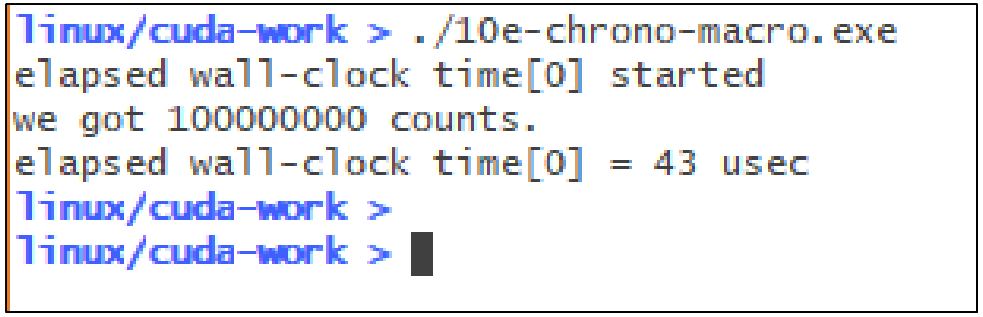
// usec = μsec = microsecond = 1 / 1,000,000 sec

in "./common.cpp"

```
chrono::system_clock::time_point __time_begin[8] = { chrono::system_clock::now(), };

#define ELAPSED_TIME_BEGIN(N)   do { \
      __time_begin[(N)] = chrono::system_clock::now(); \
      printf("elapsed wall-clock time[%d] started\n", (N)); \
      fflush(stdout); \
   } while (0)

#define ELAPSED_TIME_END(N) do { \
      chrono::system_clock::time_point time_end = chrono::system_clock::now(); \
      chrono::microseconds elapsed_msec
                        = chrono::duration_cast<chrono::microseconds>(time_end - __time_begin[(N)]); \
      printf("elapsed wall-clock time[%d] = %ld usec\n", (N), (long)elapsed_msec.count()); \
      fflush(stdout); \
   } while (0)
```

chrono-macro.cpp

```
#include "./common.cpp"

. . .

int main(void) {
   ELAPSED_TIME_BEGIN(0);
   // work
   bigJob();
   // work done
   ELAPSED_TIME_END(0);
   // done
   return 0;
}
```



```
linux/cuda-work > ./10e-chrono-macro.exe
elapsed wall-clock time[0] started
we got 100000000 counts.
elapsed wall-clock time[0] = 43 usec
linux/cuda-work >
linux/cuda-work >
```

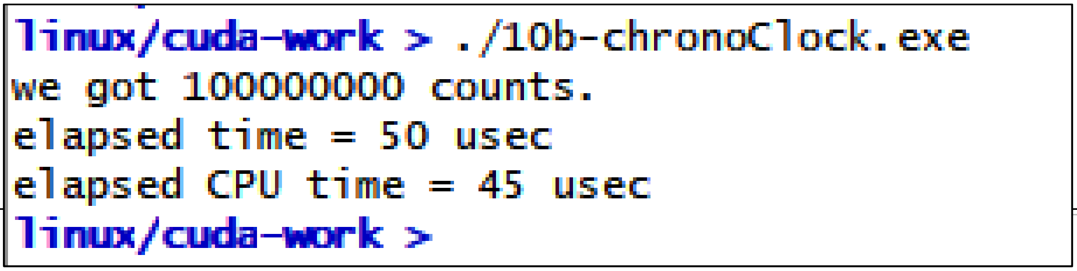
clock( ) function

- **#include <time.h>**
- **clock_t clock(void);**
  - returns an approximation of **processor time (CPU/GPU time)** used by the program
  - to get the number of seconds used, divide by CLOCKS_PER_SEC.

```
float clock_sec = (float)clock() / CLOCKS_PER_SEC;
long clock_usec = (long)(clock()) * 1000000 / CLOCKS_PER_SEC;
```

Example: chronoClock.cpp

```
int main(void) {
   system_clock::time_point chrono_begin = system_clock::now();
   clock_t clock_begin = clock();
   // work
   bigJob();
   // work done
   system_clock::time_point chrono_end = system_clock::now();
   clock_t clock_end = clock();
   // calculation
   …
   long clock_elapsed_usec = (long)(clock_end-clock_begin)*1000000/CLOCKS_PER_SEC;
   printf("elapsed CPU time = %ld usec\n", clock_elapsed_usec);
   // done
   return 0;
}
```



```
linux/cuda-work > ./10b-chronoClock.exe
we got 100000000 counts.
elapsed time = 50 usec
elapsed CPU time = 45 usec
linux/cuda-work >
```

sleep( ) function

- **pause the thread**
  - making the calling thread to sleep for the specified time periods
  - **wall-clock time 기준** (CPU time은 최소로 사용)

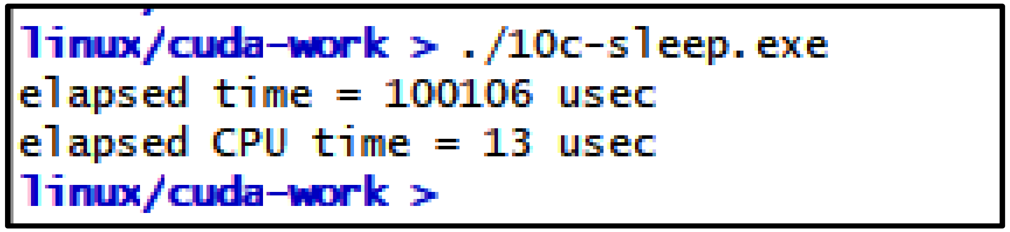
- **Unix/Linux**
  - #include <unistd.h>
  - unsinged int **sleep**(unsigned int seconds);
  - int **usleep**(useconds_t usec); // micro-seconds (1/ 1,000,000)
- **Windows**
  - #include <windows.h>
  - void **Sleep**(DWORD dwMilliseconds); // milli-seconds (1 / 1,000)

Example: sleep.cpp

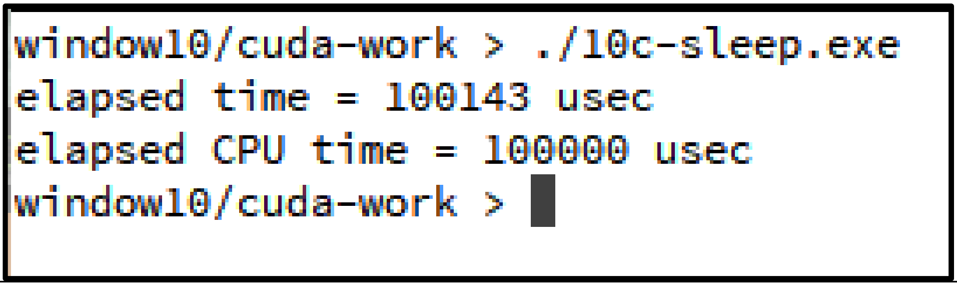
```
int main(void) {
   system_clock::time_point chrono_begin = system_clock::now();
   clock_t clock_begin = clock();
   // work
#if defined(__linux__)
   usleep(100 * 1000);  // 100 msec
#else
   Sleep(100);          // 100 msec
#endif
   // work done
   system_clock::time_point chrono_end = system_clock::now();
   clock_t clock_end = clock();
   // calculation
   ...
```

Example: sleep.cpp

```
    // calculation
    microseconds chrono_elapsed_usec
                   = duration_cast<microseconds>(chrono_end - chrono_begin);
    printf("elapsed time = %ld usec\n", (long)chrono_elapsed_usec.count());
    long clock_elapsed_usec = (long)(clock_end - clock_begin) * 1000000 / CLOCKS_PER_SEC;
    printf("elapsed CPU time = %ld usec\n", clock_elapsed_usec);
    // done
    return 0;
}
```



```
linux/cuda-work > ./10c-sleep.exe
elapsed time = 100106 usec
elapsed CPU time = 13 usec
linux/cuda-work >
```



```
window10/cuda-work > ./10c-sleep.exe
elapsed time = 100143 usec
elapsed CPU time = 100000 usec
window10/cuda-work >
```

Variable Arguments

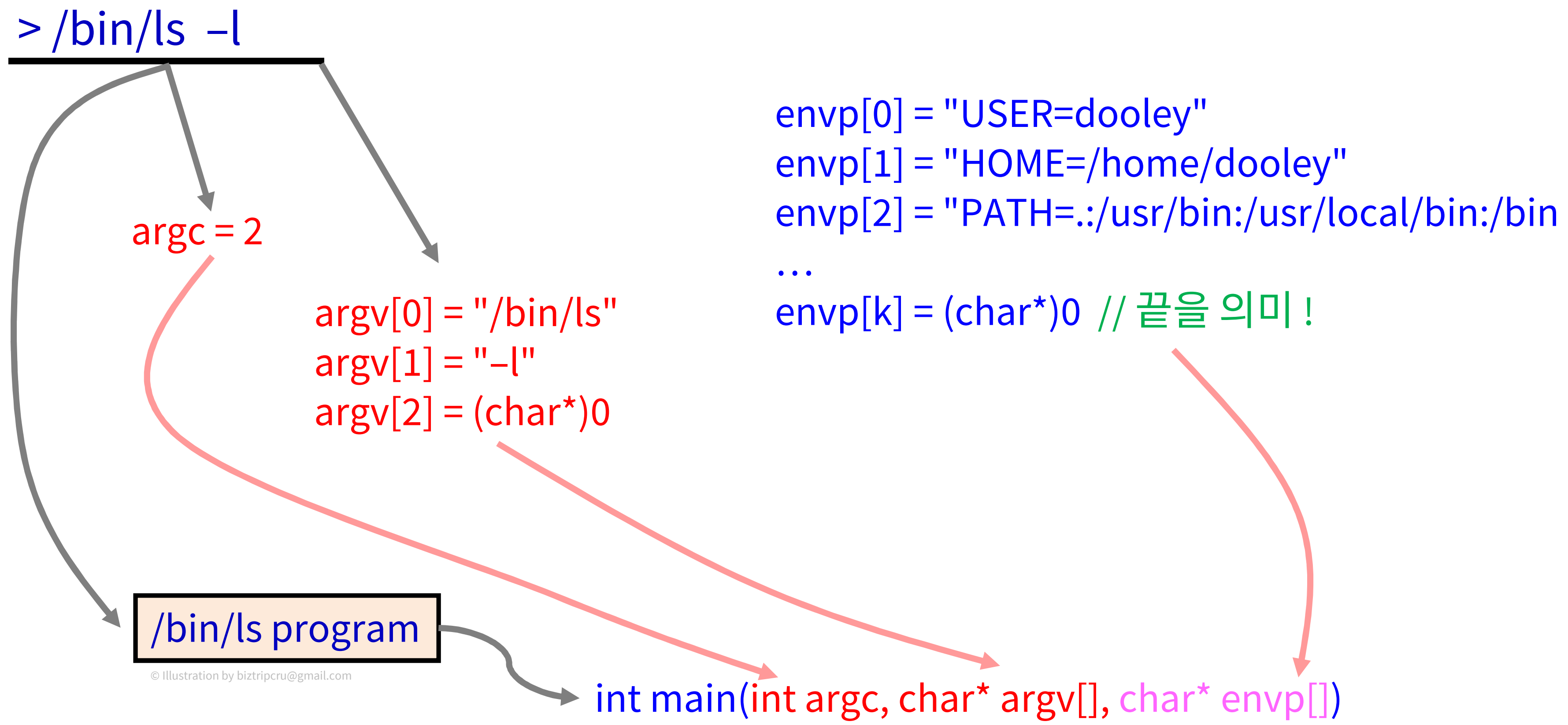
- **C/C++ main( ) 함수에서 argument를 받는 방법**
- int main(int argc, char* argv[], char* envp[]) { … }
- **명령어 창: ./a.exe alpha bravo charlie**
  - 시스템에서 자동 생성
  - argc = 4; argc = argument count
  - argv[0] = “./a.exe”; argv = argument vector (1D array)
  - argv[1] = “alpha”;
  - argv[2] = “bravo”;
  - argv[3] = “charlie”;
  - envp[…] = 환경 변수 environment variable 대입;

Environment Variables

- **environment variables**
  - shell 에서 관리하는 특별한 variables
  - 일종의 global 변수 → 모든 process가 자동으로 상속

- **PATH : an environment variable**
  - 실행 파일을 검색할 directory들을 저장
- **HOME : your home directory (or folder)**
- **USER : your user login name**

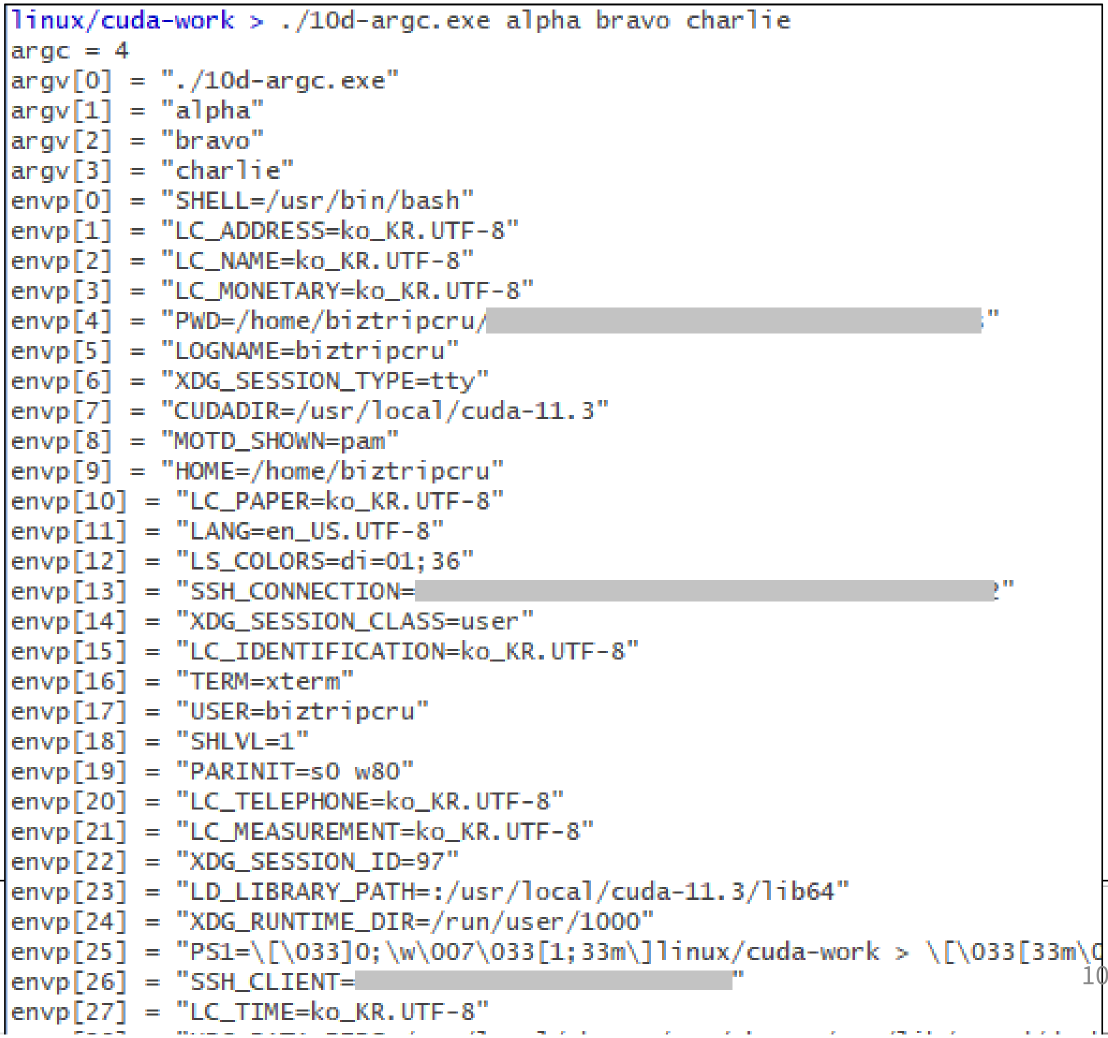
main( )에서의 해석



```
> /bin/ls  –l

argc = 2

argv[0] = "/bin/ls"
argv[1] = "–l"
argv[2] = (char*)0

envp[0] = "USER=dooley"
envp[1] = "HOME=/home/dooley"
envp[2] = "PATH=.:/usr/bin:/usr/local/bin:/bin
…
envp[k] = (char*)0  // 끝을 의미 !

/bin/ls program

int main(int argc, char* argv[], char* envp[])
```

Example: argc.cpp

```cpp
#include <stdio.h>

int main(int argc, char* argv[], char* envp[]) {
   printf("argc = %d\n", argc);
   for (int i = 0; i < argc; ++i) {
      printf("argv[%d] = \"%s\"\n", i, argv[i]);
   }
   for (int i = 0; envp[i] != nullptr; ++i) {
      printf("envp[%d] = \"%s\"\n", i, envp[i]);
   }
   // done
   return 0;
}
```



```
linux/cuda-work > ./10d-argc.exe alpha bravo charlie
argc = 4
argv[0] = "./10d-argc.exe"
argv[1] = "alpha"
argv[2] = "bravo"
argv[3] = "charlie"
envp[0] = "SHELL=/usr/bin/bash"
envp[1] = "LC_ADDRESS=ko_KR.UTF-8"
envp[2] = "LC_NAME=ko_KR.UTF-8"
envp[3] = "LC_MONETARY=ko_KR.UTF-8"
envp[4] = "PWD=/home/biztripcru/[redacted]"
envp[5] = "LOGNAME=biztripcru"
envp[6] = "XDG_SESSION_TYPE=tty"
envp[7] = "CUDADIR=/usr/local/cuda-11.3"
envp[8] = "MOTD_SHOWN=pam"
envp[9] = "HOME=/home/biztripcru"
envp[10] = "LC_PAPER=ko_KR.UTF-8"
envp[11] = "LANG=en_US.UTF-8"
envp[12] = "LS_COLORS=di=01;36"
envp[13] = "SSH_CONNECTION=[redacted]"
envp[14] = "XDG_SESSION_CLASS=user"
envp[15] = "LC_IDENTIFICATION=ko_KR.UTF-8"
envp[16] = "TERM=xterm"
envp[17] = "USER=biztripcru"
envp[18] = "SHLVL=1"
envp[19] = "PARINIT=s0 w80"
envp[20] = "LC_TELEPHONE=ko_KR.UTF-8"
envp[21] = "LC_MEASUREMENT=ko_KR.UTF-8"
envp[22] = "XDG_SESSION_ID=97"
envp[23] = "LD_LIBRARY_PATH=:/usr/local/cuda-11.3/lib64"
envp[24] = "XDG_RUNTIME_DIR=/run/user/1000"
envp[25] = "PS1=\[\033]0;\w\007\033[1;33m\]linux/cuda-work > \[\033[33m\0
envp[26] = "SSH_CLIENT=[redacted]"
envp[27] = "LC_TIME=ko_KR.UTF-8"
```

내용 contents

- Elapsed Time Calculation
- C++ Chrono features
- clock( ) function
- sleep( ) function
- argc, argv 처리

시간 측정

Elapsed Time

폰트 끝단 일치 → 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
정참판 양반댁 규수 큰 교자 타고 혼례 치른 날
본고딕 Noto Sans KR

The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
The quick brown fox jumps over the lazy dog
Source Sans Pro

Mathematical Notations O(n log n)
Source Serif Pro