Credit Suisse *2016 Sep 21*

What is the average (and the worst) time complexity of quicksort?
O(nlogn) and O(n^2) respectively.

What is the time complexity of quicksort when all elements are the same?
O(n^2), quicksort works as divide and conquer, it is the slowest when the division is always seriously uneven.

How to copy a vector of int to an array?

Deep copy :

```
std::vector<int> v;
int array[100];
std::copy(v.begin(), v.end(), array);
```

Shallow copy :

```
std::vector<int> v;
int* ptr = &v[0];
```

From vector to vector :

```
    std::vector<int> v0;
    std::vector<int> v1;
    std::copy(v0.begin(), v0.end(), std::back_inserter(v1));
//  std::copy(v0.begin(), v0.end(), v1.begin()); // incorrect
```

What are the differences among const cast, static cast, dynamic cast and reinterpret cast?
All castings convert an expression into new type, using the syntax : `xxx_cast<new_type>(expression)`. Const cast adds or removes const-ness to (from) expression, it is used when 3rd party API that does not treat const-ness properly. Static cast perform type conversion in 2 ways : (1) implicit conversion (invoked by function call or operator, which accepts new type as argument, but not expression type) and (2) explicit conversion (invoked by `static_cast<new_type>()`), both conversions are executed either by new type's conversion constructor (constructor with one argument) or expression type's conversion operator into the new type (what is the precedence?). Implicit conversion can be forbidden by adding `explicit` before constructor. Dynamic cast can only used with pointer to objects of the same inheritance hierarchy, we can either upcast (to base class) or downcast (to derived class). Reinterpret cast can convert pointer to a type to pointer to any other type, this is dangerous and rarely used. It is usually used in when interfacing with vender's API over which we cannot control : reinterpret cast pointer to my class into vender's class, invoke vender's API, and reinterpret cast it back to my class. Hence const cast handles const-ness, static cast handles object, while the other two casts handle pointers. However, static cast can be used to handle pointer too. Oh, how? What is the precedence for static cast from class A to class B, suppose we have :

```
A::operator B()
B::B(const A&)
A object_A;
B object_B0 = A();                 // implicit static cast, will invoke A::operator B()
B object_B1 = static_cast<B>(A);   // expliciy static cast, will invoke (please check)
```

In the above example, implicit static cast invokes conversion operator (rather than conversion constructor), since there is a rule : use the one with less const qualification. Conversion constructor will be called when we add const-ness to class A's conversion operator : `A::operator B() const`. Please read http://www.cplusplus.com/doc/tutorial/typecasting/

What do the keyword static and volatile mean?
Static member belongs to a class (rather than belongs to a particular object, thus do not use `this` pointer inside static member function). We can invoke static member function without declaring any object. It is like a standalone function within the class scope, thus it serves as a global helper.

Keyword volatile is used with IO and multithread. When we are reading a spot in memory, that is continuously updated by another process (or device), the compiler thinks that the variable may be useless (as it does not change, in compiler's point of view), then the code optimizer may make some undesirable change to the code, resulting in logic error. Here is an example :

```
volatile unsigned long* ptr_accessed_by_another_process = initial_address;
while(*ptr_accessed_by_another_process != ok_to_proceed) do_something;
```

Why do we use smart pointer? What are the differences among different smart pointers?
Smart pointers can be classified into : (1) no ownership, (2) local ownership, (3) transfer of ownership and (4) shared ownership. Type 1 means that the smart pointer doesn't own the resource and therefore cannot free the resource. Type 2 is for RAII purpose, it cannot transfer the ownership, cannot pass it anywhere, cannot return from functions, it frees the resource once the smart pointer runs out of scope. Type 3 smart pointer cannot transfer ownership by copy constructor nor by assignment operator, in fact it should be done by moving constructor and moving assignment operator, which are supported in C++1X, unique_ptr is an example. Type 4 smart pointer can be done by copy constructor and assignment operator, typical example is the shared_ptr, which keeps a reference count.

type 1	`weak_ptr`	non-owning , used as a reference to shared_ptr without adding reference count
type 2	`scoped_ptr`	neither transferable nor sharable, freed when out of scope
type 3	`unique_ptr`	we cannot copy it, but we can move it by using C++1x move constructors
type 4	`shared_ptr`	

Not 100% correct

`weak_ptr` provides reference to resource managed by `shared_ptr`. If you need to access the resource, you can lock the management of it (avoid `shared_ptr` in another thread frees it while you use the `weak_ptr`) and use it. If weak_ptr points to something already deleted, it will throw an exception. Using weak_ptr is most beneficial when you cyclic reference of `shared_ptr`, which has incorrect reference counting. `intrusive_ptr` is like a `shared_ptr` but it delegates reference count to some helper functions that need to be defined by the object being managed. The semantic (interface) of `auto_ptr` is the same as `unique_ptr`, but because of missing native support for moving, `auto_ptr` fails to provide moving functionality without pitfalls. `auto_ptr` will be deprecated in next C++ standard release in favor of `unique_ptr`.

What is a spin lock? What is a mutex lock?
When a thread tries to lock a spinlock and if it fails (as it is already locked), it will continuously retry locking it, until it finally succeeds. But when a thread tries to lock a mutex and if it fails (as it is already locked), it will go to sleep until being woken up, which will be the case once the mutex is being unlocked by whatever thread holding the lock before. There is problem with both spinlock and mutex. Polling on a spinlock constantly wastes CPU time. On the other hand, mutex puts threads to sleep and wakes them up, both are rather expensive operations. If mutex locks for a very short amount of time, the time spent in putting a thread to sleep and waking it up again might exceed the time the thread has actually slept by far and it might even exceed the time the thread would have wasted by constantly polling on spinlock.

[Solution]
For single core system, spinlock doesn't make sense, as the thread polling spinlock is blocked and withdraw all CPU time from other threads, no other thread can run, and since no other thread can run, the lock wont be unlocked either. In this case, we need to use mutex and put the thread to sleep.

For multi-core system, with plenty of locks that are held for a very short amount of time only, time wasted in constantly putting threads to sleep and waking them up again might decrease runtime performance noticeably. Thus its better to use spinlock, threads are blocked for a very short period, but then immediately continue their work, leading to higher processing throughput.

Not 100% correct

A hybrid mutex behaves like a spinlock at first on a multi-core system. If a thread cannot lock the mutex, it won't be put to sleep immediately, since the mutex might get unlocked pretty soon. Only if the lock has still not been obtained after a certain amount of time (or retries or any other measuring factor), the thread is really put to sleep. If the same code runs on a system with only a single core, the mutex will not spinlock.

If a program consumes 100% CPU, what will you do?
In linux, CPU utilization can be checked by command "top", then what? Does log help? Does heartbeat help? Should I kill the process? Is there high waiting on IO? Is there high interrupt processing (a process that is issuing lots of software interrupts).

Do you know CPU affinity / cache missing / context switching / kernel crossing?
CPU affinity enables binding and unbinding of a process or a thread to a CPU or multiple CPUs, so that the process will execute only on the designated CPU or CPUs rather than any CPU. This can be viewed as a modification of the central queue scheduling algorithm in a OS. In windows, changing affinity and priority is easy, bring up the task manager, right click the process you need. In linux, please use command "taskset", given process id (pid), we will get a CPU mask, i.e. an array of binary flags indicating whether the corresponding CPU is allowed to execute the process, for example, mask 0x0003 means a process run by CPU#0 and CPU#1.

Not 100% correct

Cache is a piece of smaller and faster memory, which stores a copy of frequently used data from the main memory. A cache miss refers to a failed attempt to read or write a piece of data in the cache, which results in a main memory access with much longer latency. There are 3 kinds of cache misses : instruction read miss, data read miss, and data write miss.

In order to lower cache miss rate, a great deal of analysis has been done on cache behavior in an attempt to find the best combination of size, associativity, block size. Please read https://en.wikipedia.org/wiki/CPU_cache#CACHE-MISS

Context switch is an essential feature of a multitasking operating system which enables multiple processes to share a single CPU. When it switches one process out of a CPU and switches another process into the CPU, it invokes a process of storing out-process's state and restoring in-process's state, so that execution of out-process can be resumed from the same point at a later time. Yet switching is expensive. See "Effective Modern C++" by Scott Meyers, item35.

What are Waterfall, Agile methodology, Test Driven Development (TDD) and pair programming?
Waterfall methodology is a traditional software development, it is sequential, it emphasizes well planning and execution sticking to plan. Its workflow is : specification drafting, software design, implementation, verification and maintainence. Developers need to predict a far future, predict what the deliverable should be. However, prediction may be wrong, the team may see a different picture, or a changing user requirement as the project proceeds. Waterfall lacks of flexibility, besides the sunk cost is expensive (when the team finally find out the deliverable is not what user want when it is 90% completed).

Agile methodology is in fact a philosophy, mindset, set of values, a practice or habit, requiring strong self-discipline. Its main idea is to break the project down into phases, like 4 times 25%. Each time, we plan and execute 25% of it, we don't plan too far in the future (as we never predict correctly). Furthermore, it emphasizes on : (1) responding to change over sticking to a plan (it welcomes changes in user specification, even late changes), (2) frequent delivery (as a measure of progress), working on software over comprehensive documentation, (3) individuals and interactions over processes and tools, collaboration among team members, frequent feedback, face to face communication. If waterwall is a predictive methodology, then Agile is on the adaptive side of ideology. Waterfall is good for large number junior developers who demand order, while Agile works for small number of senior devlopers who response to changes.

TDD, as its name implies, is test oriented. Test serves at least 3 purposes : (1) definition of user specification, (2) a proof that current version software fails to meet user specification, and therefore needs an enhancement, thus developers need to write code, (3) a proof that current version software fulfills requirement and is ready to rollout. TDD turns traditional mindset upside down, TDD followers refuse to write code until there exists a test proving that enhancement is needed, TDD followers write code that just enough to pass the test (they don't overdesign anything). New tests can be added when current software passes all existing tests, besides, each test focuses on one or a few functionalities, don't write a huge test. The act of testing is in fact the act of design.

Pair programming : one driver and one navigator, they switch their roles regularly.