الباب الرابع

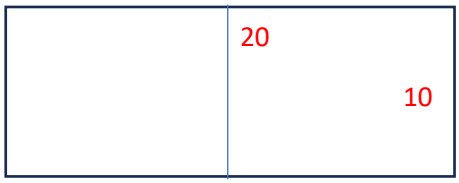
الترتيب السريع (Quicksort):

فـي الفصل السابق تحدثنا عن ال recursionو في هذ الفصل سنتحدث عن ألجوريزم يطبق ال recursion و هو ال divide and conquer و هو ما سنستخدمه لتنفيذ ال quicksort.

فرق تسد (divide and conquer):

لنفترض أن لديك ورقة مستطيلة الشكل و تريد تقسيمها إلى مربعات متساوية على أن يكون المربع كبيرا قدر المستطاع , يمكننا حل هذه المشكلة باستخدام divide and conquer

أولا ال base case التي سنتوقف حين نصل إليها أن يكون طول أحد الأضلاع من مضاعفات الآخر و يكون هو النتيجة



ال recursive case نقسم المستطيل إلى مربعات ضلعها يساوي الضلع الأصغر في المستطيل , و من ثم سيتبقى مستطيل أصغر سنقوم بتنفيذ العملية السابقة عليه حتى نصل إلى ال base case.

من المسال السابق يمكننا أن نفهم أن ال divide and conquer تقسم المشكلة الكبيرة إلى مشكلة أصغر يسهل الوصول من خلالها إلى الحل

مثال آخر , نريد مجموع العناصر الموجودة في هذا ال array {2,4,6}
يمكننا بالطبع أن نجد الحل بسهولة من خلال ال loop و لكن ماذا عن ال recursion

سنقسم مشكلتنا إلى مشاكل أصغر كالتالي:

1. sum({2,4,6})-> 12
2. 2+sum({4,6})-> 2+10=12
3. 4+sum({6})->4+6= 10
4. 6+sum({})->empty is the base case that returns 0 , so it returns 6

خلي بالك ال stack شغال last in first out فأول function عي آخر واحدة هتخلص.

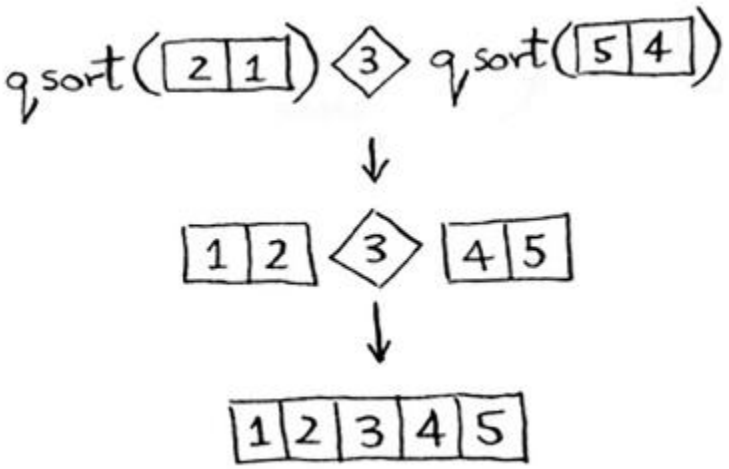
```
template <typename T>
T sum(std::vector<T> arr) {
    if (arr.empty()) return 0;

    T last_num = arr.back(); // save last number value
    arr.pop_back();  // and remove it from array for next recursive call
    return last_num + sum(arr);
}
```

الترتيب السريع (Quicksort):

-ال base case عندما يكون ال array فيه عنصر واحد أو فارغ لذا لا يحتاج إلى ترتيب
-إذا كان هناك عنصران نقوم بالتبديل إذا لم يكونا مرتبين و لا نفعل شيئا لو كانا مرتبين

-ماذا إذا كانت العناصر أكثر؟ سنختار أحد العناصر و ليكن العنصر الأول و سنسميه ال pivot بعد ذلك سنضم العناصر الأكبر في array آخر و نفس الأمر مع العناصر الأصغر , هكذا يصبح لدينا 2 arrays بجانب ال pivot و نقوم بنفس الخطوات مع كل منهم حتى نصل إلى ال base case و ندمج العناصر مجددا حيث يكون ال pivot في المنتصف



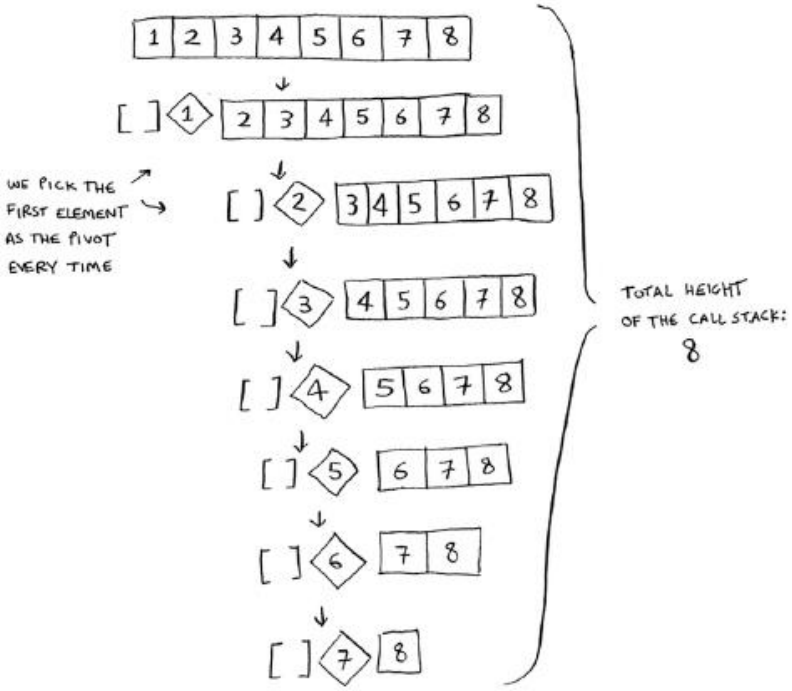
في المتوسط يحتاج ال quicksort إلى O(n*log(n)) بينما في أسوأ حالة O(n^2)

هناك نوع من الترتيب و هو ال merge sort يحتاج إلى O(n*log(n)) دائما , هل هذا يعني أنه أفضل؟ لماذا لا نستخدمه بدلا من ال quicksort؟

لو أردنا أن نطبع عدد n من العناصر علما بأن طباعة كل عنصر يحتاج إلى c من الوقت في هذه الحالة فإننا نحتاج إلى c*n وقت لإتمام العملية في O(n) لاحظ أننا نتجاهل الرقم الثابت

c لأنه لا يحدث تأثير في أغلب الحالات و لكن في حالة ال quicksort و ال merge sort فإن هذا الثابت يعطي أفضلية للquicksort خاصة و أنها تحتاج إلى (O(n*log(n)) فقط في غالب الحالات.

تعتمد الحالة المتوسطة أو الحالة الأسوأ على اختيارك لل pivot لاحظ الفرق:



في هذه الحالة فإن طول ال stack هو نفس عدد العناصر n , لدينا n من المستويات و كل مستوى يحتاج إلى O(n) لذا أسوأ حالة O(n*n) أو O(n^2)

في الحالة المتوسطة (أغلب الحالات) يكون طول ال stack هو log(n) فقط لذا نحتاج O(n*log(n)) كما في الصورة التالية: