Comments

- Rand is a costing function
- Initialization should be removed from the timing
- Array with distinct values
- Redundant data allocation
- Rand_select with no randomization
- Select return the value instead of the position

Rand function

```
int randPartition(std::vector<int>& arr, int p, int q) {
   std::random_device randomDevice;
   std::mt19937 generator(randomDevice());
   std::uniform_int_distribution<> distr(p, q);
   int random_index = distr(generator);
   return partition(arr, p, q, random_index);
}
```

Sorting…

```
for (auto _ : state()) {
  benchmark::DoNotOptimize(run_select_ith(index, &arr, 0, n));
}
///
for (auto _ : state()) {
  benchmark::DoNotOptimize(run_select_ith(index, &arr, 0, n));
  // Shuffle array
  state.PauseTiming();
  std::shuffle(arr.begin(), arr.end(), g);
  state.ResumeTiming();
}
```

Array with distinct values

- Adopted to simplify the pivot search

Data allocation

```
Pair Median(vector<Pair>& A, int start, int end) {
    int size = end - start + 1;
    vector<Pair> temp(A.begin() + start, A.begin() + end + 1);
    insertionSort(temp);
```

Passing the data by reference is the efficient way.

Avoid redundant copies.

Avoid redundant copies for the medians

- Move the medians at the beginning

```
for (int i = 0; i < arr_len / GROUP_LEN; ++i) {
  bubbleSort(values, start + i * GROUP_LEN, GROUP_LEN);
  /* put the medians to the first several position
    (no need to allocate more space to store medians) */
  std::swap(values[start + i], values[start + i * GROUP_LEN + (GROUP_LEN - 1) / 2]);
}
```

No random partitioning

```
int partition(std::vector<int>& array, int p, int q)
{
    int x = array[p]; // The pivot is the first element
    int i = p;
    for (int j = p + 1; j <= q; j++) {
        if (array[j] < x) {
            i++;
            std::swap(array[i], array[j]);
        }
    }
    std::swap(array[p], array[i]);
    return i;
}
```