Engineering Faster Sorters for Small Sets of Elements

Jasper Marianczuk, Timo Bingmann

Idee für eine Bachelorarbeit

```
template <typename TValueType>
static inline
void InsertionSort(TValueType* items, size_t arraySize)
{
  int inner, outer;
  for (outer = 1; outer < arraySize; outer += 1) {
    TValueType current = items[outer];
    for (inner = outer; inner > 0 && items[inner - 1] > current; inner -= 1)
    {
      items[inner] = items[inner - 1];
    }
    items[inner] = current;
  }
}

template<>
inline
void InsertionSort<SortableRef_StlVersion>(SortableRef_StlVersion* first, size_t arraySize
{
  SortableRef_StlVersion* last = first + arraySize - 1;
  for (SortableRef_StlVersion* next = first; ++next != last; )
  {
    SortableRef_StlVersion* next_temp = next;
    SortableRef_StlVersion val = *next;

    if (val < *first)
    {
      next_temp++;
      while (first != next)
      {
        *--next_temp = *--next;
      }
      *first = val;
    }
    else
    {
      for (SortableRef_StlVersion* first_temp = next_temp;
           val < *--first_temp;
           next_temp = first_temp)
      {
        *next_temp = *first_temp;
      }
      *next_temp = val;
    }
  }
}
```

```
template <typename TValueType>
static inline
void ConditionalSwap(TValueType& left, TValueType& right)
{
  if (left > right) {std::swap(left, right); }
}

template<>
inline
void ConditionalSwap<Sortable_JumpXchg>(Sortable_JumpXchg& left, Sortable_JumpXchg& right)
{
  __asm__(
    "cmpq %[left_key],%[right_key]\n\t"
    "jae %=f\n\t"
    "xchg %[left_key],%[right_key]\n\t"
    "%=:\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key)
    :
    : "cc"
  );
}

template<>
inline
void ConditionalSwap<SortableRef_JumpXchg>(SortableRef_JumpXchg& left, SortableRef_JumpXch
{
  __asm__(
    "cmpq %[left_key],%[right_key]\n\t"
    "jae %=f\n\t"
    "xchg %[left_key],%[right_key]\n\t"
    "xchg %[left_reference],%[right_reference]\n\t"
    "%=:\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key),
      [left_reference] "+r"(left.reference), [right_reference] "+r"(right.reference)
    :
    : "cc"
  );
}
```

```
template<>
inline
void ConditionalSwap<Sortable_TwoCmovTemp>(Sortable_TwoCmovTemp& left, Sortable_TwoCmovTem
{
  uint64_t tmp = left.key;
  __asm__(
    "cmpq %[left_key],%[right_key]\n\t"
    "cmovbq %[right_key],%[left_key]\n\t"
    "cmovbq %[tmp],%[right_key]\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key)
    : [tmp] "r"(tmp)
    : "cc"
  );
}

template<>
inline
void ConditionalSwap<SortableRef_FourCmovTemp>(SortableRef_FourCmovTemp& left, SortableRef
{
  uint64_t tmp = left.key;
  uint64_t tmpRef = left.reference;
  __asm__(
    "cmpq %[left_key],%[right_key]\n\t"
    "cmovbq %[right_key],%[left_key]\n\t"
    "cmovbq %[right_reference],%[left_reference]\n\t"
    "cmovbq %[tmp],%[right_key]\n\t"
    "cmovbq %[tmp_ref],%[right_reference]\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key),
      [left_reference] "+r"(left.reference), [right_reference] "+r"(right.reference)
    : [tmp] "r"(tmp), [tmp_ref] "r"(tmpRef)
    : "cc"
  );
}
```

```
template<>
inline
void ConditionalSwap<SortableRef_FourCmovTemp_Split>(SortableRef_FourCmovTemp_Split& left,
{
  uint64_t tmp = left.key;
  uint64_t tmpRef = left.reference;
  __asm__(
    "cmpq %[left_key],%[right_key]\n\t"
    :
    : [left_key] "r"(left.key), [right_key] "r"(right.key)
    : "cc"
  );
  __asm__(
    "cmovbq %[right_key],%[left_key]\n\t"
    : [left_key] "+r"(left.key)
    : [right_key] "r"(right.key)
    :
  );
  __asm__(
    "cmovbq %[right_reference],%[left_reference]\n\t"
    : [left_reference] "+r"(left.reference)
    : [right_reference] "r"(right.reference)
    :
  );
  __asm__(
    "cmovbq %[tmp],%[right_key]\n\t"
    : [right_key] "+r"(right.key)
    : [tmp] "r"(tmp)
    :
  );
  __asm__(
    "cmovbq %[tmp_ref],%[right_reference]\n\t"
    : [right_reference] "+r"(right.reference)
    : [tmp_ref] "r"(tmpRef)
    :
  );
}
```

```
template<>
inline
void ConditionalSwap<Sortable_ThreeCmovRegisterTemp>(Sortable_ThreeCmovRegisterTemp& left,
{
  register uint64_t tmp;
  __asm__ (
    "cmpq %[left_key],%[right_key]\n\t"
    "cmovbq %[left_key],%[tmp]\n\t"
    "cmovbq %[right_key],%[left_key]\n\t"
    "cmovbq %[tmp],%[right_key]\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key), [tmp] "+r"(tmp)
    :
    : "cc"
  );
}

template<>
inline
void ConditionalSwap<SortableRef_SixCmovRegisterTemp>(SortableRef_SixCmovRegisterTemp& lef
{
  register uint64_t tmp;
  register uint64_t tmpRef;
  __asm__ (
    "cmpq %[left_key],%[right_key]\n\t"
    "cmovbq %[left_key],%[tmp]\n\t"
    "cmovbq %[left_reference],%[tmp_ref]\n\t"
    "cmovbq %[right_key],%[left_key]\n\t"
    "cmovbq %[right_reference],%[left_reference]\n\t"
    "cmovbq %[tmp],%[right_key]\n\t"
    "cmovbq %[tmp_ref],%[right_reference]\n\t"
    : [left_key] "+r"(left.key), [right_key] "+r"(right.key),
      [left_reference] "+r"(left.reference), [right_reference] "+r"(right.reference),
      [tmp] "+r"(tmp), [tmp_ref] "+r"(tmpRef)
    :
    : "cc"
  );
}
```

```
template<>
inline
void ConditionalSwap<SortableRef_ClangVersion>(SortableRef_ClangVersion& left, SortableRef
{
  register SortableRef_ClangVersion* leftPointer = &left;
  register SortableRef_ClangVersion* rightPointer = &right;
  register uint64_t rightKey = right.key;
  SortableRef_ClangVersion tmp = left;
  __asm__(
    "cmpq %[tmp_key],%[right_key]\n\t"
    "cmovbq %[right_pointer],%[left_pointer]\n\t"
    : [left_pointer] "+r"(leftPointer)
    : [right_pointer] "r"(rightPointer),
      [tmp_key] "m"(tmp.key), [right_key] "r"(rightKey)
    : "cc"
  );
  left = *leftPointer;
  leftPointer = &tmp;
  __asm__(
    "cmovbq %[left_pointer],%[right_pointer]\n\t"
    : [right_pointer] "+r"(rightPointer)
    : [left_pointer] "m"(leftPointer)
    :
  );
  right = *rightPointer;
}
```